Query evaluation via tree-decompositions

Jörg Flum   Markus Frick   Martin Grohe
Institut für Mathematische Logik, Eckerstr. 1, 79104 Freiburg, Germany
Email: flum@uni-freiburg.de, {frick,grohe}@logik.mathematik.uni-freiburg.de

May 17, 2000

Abstract

A number of efficient methods for evaluating first-order and monadic-second order queries on finite relational structures are based on tree-decompositions of structures or queries. We systematically study these methods.

In the first-part of the paper we consider arbitrary formulas on tree-like structures. We generalize a theorem of Courcelle [8] by showing that on structures of bounded tree-width a monadic second-order formula (with free first- and second-order variables) can be evaluated in time linear in the structure size plus the size of the output.

In the second part we study tree-like formulas on arbitrary structures. We generalize the notions of acyclicity and bounded tree-width from conjunctive queries to arbitrary first-order formulas in a straightforward way and analyze the complexity of evaluating formulas of these fragments. Moreover, we show that the acyclic and bounded tree-width fragments have the same expressive power as the well-known guarded fragment and the finite-variable fragments of first-order logic, respectively.

1 Introduction

Evaluating first-order, or relational calculus, queries against a finite relational database is well-known to be PSPACE-complete [16]. The complexity we refer to here is called the *combined complexity* of the query language [20], i.e. the complexity of the evaluation problem is measured both in terms of the length of the query and the size of the database. Many research efforts went into handling this high worst case complexity. In practice, various query optimization heuristics are used; they are based both on the structure of the queries and the (expected) structure of the databases.

One of the important theoretical notions is that of *acyclic conjunctive queries* (cf. [1]). Yannakakis [22] proved that acyclic conjunctive queries can be recognized and evaluated efficiently. In the last few years, there has been renewed interest in acyclic conjunctive queries and related notions based on the graph theoretic concept of *tree-width* [7, 13, 11, 12]. Whereas these approaches concentrate on conjunctive queries, there have also been attempts to isolate larger fragments of first-order logic whose combined complexity is in PTIME. In particular, Vardi [21] observed that this is the case for the *finite variable fragments* of first-order logic. A fragment of first-order logic that has recently received a lot of attention is the *guarded fragment* [3]. Gottlob, Grädel, and Veith [10] proved that its combined complexity is both linear in the length of the formula and the size of the database.

There is a different way of using tree-width to evaluate queries that originated in the area of graph algorithms. Here we do not restrict the class of queries, but the class of input structures, or databases. This approach does not only work for first-order logic, but even for the much stronger monadic second-order logic. Courcelle [8] proved that Boolean monadic second-order queries on structures of bounded tree-width can be evaluated in time linear in the size of the input structure, or more precisely in time $f(l) \cdot n$, where l is the length of the query, n is the size of the structure, and $f : \mathbb{N} \to \mathbb{N}$ is some (fast-growing) function.[1] Arnborg, Lagergren, and Seese [5] extended Courcelle's result by showing that on structures of bounded tree-width the number of satisfying assignments of a monadic second-order formula with free variables can be computed in time linear in the size of the input structure.

[1] Another way to phrase Courcelle's result is to say that the *data complexity* [20] of monadic second-order logic on graphs of bounded tree-width is in linear time. However, in this paper we consequently maintain the view of combined complexity.

In the first part of this paper we shall further extend this approach by proving that on structures of bounded tree-width the set of all satisfying assignments of a monadic second-order formula (with free first and second-order variables) can be computed in time linear in the size of the input structure and the size of the output. For example, *all* cliques of a graph $\mathcal{G}$ of bounded tree-width can be computed in time $O(|G| + \sum_{C \text{ clique in } \mathcal{G}} |C|)$. Similarly, all Hamiltonian cycles of a graph of bounded tree-width can be computed in time linear in the size of the graph and the output.

In the second part of the paper we study fragments of first-order logic whose formulas have a tree-like structure. We start by reviewing Yannakakis's [22] algorithm for evaluating acyclic conjunctive queries. Practically the same algorithm can be used to evaluate conjunctive queries of bounded tree-width [7, 13], bounded query-width [7], or bounded hypertree-width [12]. We then extend the notions of acyclicity and tree-width from conjunctive queries to full first-order logic. Our approach is based on the well-known correspondence between first-order formulas and non-recursive stratified datalog programs (cf. [1]). We generalize acyclicity and tree-width in a straightforward way to such datalog programs. Yannakakis's algorithm immediately gives us algorithms for evaluating queries defined by acyclic or bounded-tree-width programs.

Then we show that the tree-like fragments of first-order logic obtained by this approach are very closely related to the two fragments mentioned earlier: We show that acyclic programs correspond to guarded first-order formulas and bounded tree-width programs correspond to finite-variable first-order formulas. The latter extends an observation Kolaitis and Vardi [13] made on the level of conjunctive queries. Let us remark that this correspondence is only true for sentences and for formulas with "few" free variables (a formula in the k-variable fragment of first-order logic can have at most k free variables, whereas a formula of tree-width k can have arbitrarily many free variables). In particular the result on the guarded fragment, though not difficult to prove, is quite remarkable, since the motivation for introducing the guarded fragment was completely different. Nevertheless, it turns out that this new fragment can be described in terms of the well-known concept of acyclicity.

The second part of our paper thus shows that Yannakakis's method of evaluating tree-like queries is very far reaching and comprises all the methods used to evaluate queries in the other fragments of first-order logic mentioned here.

In the last section we discuss the connections between the two parts of the paper. The algorithms in both parts are very similar, and we give an explanation of why that is using tree-automata.

Throughout the paper, it is one of our main objectives to analyze the running times of the algorithms as precisely as possible, instead of just saying that we have polynomial time algorithms. Occasionally, this requires considerable additional efforts. We have added an appendix in which we show how to implement some basic algorithmic routines, such as computing joins of two relations in linear time (in the size of the input and output).

Furthermore, we are always interested in evaluating formulas with free-variables and not just sentences. Let us also emphasize that we are not fixing a vocabulary for our formulas and structures in advance, but let the vocabulary vary with the inputs.

2 Preliminaries

Structures and Queries. A *vocabulary* is a finite set of relation symbols. Associated with every relation symbol is a natural number, its *arity*. The arity of a vocabulary is the maximum of the arities of the relation symbols it contains. In the following, τ always denotes a vocabulary.

A τ*-structure* $\mathcal{A}$ consists of a non-empty set A, called the *universe* of $\mathcal{A}$, and a relation $R^{\mathcal{A}} \subseteq A^r$ for each r-ary relation symbol $R \in \tau$. If $\mathcal{A}$ is a structure and $B \subseteq A$ non-empty, then $\langle B \rangle^{\mathcal{A}}$ denotes the substructure induced by $\mathcal{A}$ on B.

We only consider finite structures. It is convenient to assume that elements of a structure are natural numbers. In other words, *the universe of every structure considered here is a finite subset of* $\mathbb{N}$. Let us remark that all the results of this paper remain true if we also admit constants in our structures. We restrict our attention to the relational case because constants would not give us additional insights.

STR denotes the class of all structures (whose universes are finite subsets of $\mathbb{N}$). If C is a class of structures, $C[\tau]$ denotes the subclass of all τ-structures in C. We consider graphs as $\{E\}$-structures

$\mathcal{G} = (G, E^{\mathcal{G}})$, where $E^{\mathcal{G}}$ is an anti-reflexive and symmetric binary relation (i.e. graphs are loop-free and undirected). A *colored graph* is a structure $\mathcal{C} = (C, E^{\mathcal{C}}, P_1^{\mathcal{C}}, \ldots, P_n^{\mathcal{C}})$, where $(C, E^{\mathcal{C}})$ is a graph and the unary relations $P_1^{\mathcal{C}}, \ldots, P_n^{\mathcal{C}}$ form a partition of the universe C.

A *k-ary query* of vocabulary τ is a mapping χ that associates with each structure $\mathcal{A} \in \mathrm{STR}[\tau]$ a k-ary relation $\chi(\mathcal{A}) \subseteq A^k$ in such a way that for every isomorphism $f : \mathcal{A} \to \mathcal{B}$ between structures $\mathcal{A}, \mathcal{B} \in \mathrm{STR}[\tau]$ we have $\chi(\mathcal{B}) = f(\chi(\mathcal{A}))$. We admit $k = 0$ and let A^0 consist of one element (the empty tuple) for every A. We identify A^0 with TRUE and $\emptyset$ with FALSE. 0-ary queries are usually called Boolean queries.

Logics. We assume that the reader is familiar with first-order logic and monadic second-order logic. FO and MSO denote the classes of first-order and monadic second-order formulas, respectively. If L is a class of formulas, then $\mathrm{L}[\tau]$ denotes the class of all formulas of vocabulary τ in L.

We always use lowercase letters $x, y, \ldots$ to denote first-order variables and uppercase letters $X, Y, \ldots$ to denote monadic second-order variables. The set of free variables of a formula φ is denoted by free(φ). We write $\varphi(X_1, \ldots, X_l, x_1, \ldots, x_m)$ to indicate that free$(\varphi) = \{X_1, \ldots, X_l, x_1, \ldots, x_m\}$. However, if we just write φ this does not mean that φ is a sentence, i.e. that free$(\varphi) = \emptyset$.

For a structure $\mathcal{A} \in \mathrm{STR}$ and a formula $\varphi(X_1, \ldots, X_l, x_1, \ldots, x_m)$ we let

$$\varphi(\mathcal{A}) := \big\{(A_1, \ldots, A_l, a_1, \ldots, a_m) \mid \mathcal{A} \models \varphi(A_1, \ldots, A_l, a_1, \ldots, a_m)\big\}.$$

For sentences we have $\varphi(\mathcal{A}) =$ TRUE if, and only if, $\mathcal{A}$ satisfies φ. If φ has no free second-order variables, then we call the mapping $\mathcal{A} \mapsto \varphi(\mathcal{A})$ the query *defined* by φ.

In this paper, for various logics L and classes C of structures we will study the following *evaluation problem for* L *on* C:

Input:	Structure $\mathcal{A} \in C$, formula $\varphi \in \mathrm{L}$.
Problem:	Compute $\varphi(\mathcal{A})$.

If C is the class of all structures, then we call this problem the *evaluation problem for* L. Note that the input structures and formulas in the evaluation problem are of varying vocabulary. If the vocabularies of $\mathcal{A}$ and φ are different, then we have $\varphi(\mathcal{A}) = \emptyset$.

We often denote tuples $(a_1, \ldots, a_k)$ of elements of a set A by $\bar{a}$, and we write $\bar{a} \in A$ instead of $\bar{a} \in A^k$. Similarly, we denote tuples of $(B_1, \ldots, B_k)$ of subsets of A by $\bar{B}$ and write $\bar{B} \subseteq A$ instead of $\bar{B} \in (\mathrm{Pow}(A))^k$. We denote tuples of variables by $\bar{x}$ or $\bar{X}$.

Coding issues. In finite model theory, one usually assumes that structures are coded by $\{0,1\}$-strings in a way that generalizes the adjacency matrix encoding of graphs. This is appropriate for the complexity theoretic results that form the core of finite model theory, but not for the more algorithmic considerations of this paper. Since in the finite model theory literature this issue is usually not addressed, we think it is worthwhile to explain our encoding and some basic algorithmic routines in some detail.

Our underlying model of computation is the standard RAM-model with addition and subtraction as arithmetic operations (cf. [2, 19]). As common, we assume that initially all memory registers are set to '0'. In our complexity analysis we use the uniform cost measure.

We will carefully distinguish between the *size* $||o||$ of an object o, which is the length of its encoding, and, if o is a set, its *cardinality*, denoted by $|o|$. For example, if R is an r-ary relation on a set A, then for a reasonable encoding of R we will have $||R|| = O(r \cdot |R| + 1)$.

To represent structures on a RAM, we first have to encode the vocabularies. We assume that relation symbols are just pairs of natural numbers, (i, j) is the ith j-ary relation symbol. The encoding $[\tau]$ of τ is a sequence of $2|\tau| + 1$ natural numbers. The first is $|\tau|$, it is followed by all the pairs $(i, j) \in \tau$. Then $||\tau|| = 1 + 2|\tau|$ is the length of the encoding of τ.

The encoding of a structure $\mathcal{A} \in \mathrm{STR}[\tau]$ is also a sequence $[\mathcal{A}]$ of natural numbers. It starts with the encoding of the vocabulary. This is followed by the size of the universe and the sizes of all relations in $\mathcal{A}$.

The next $|A|$ numbers are the elements of the universe. After that the relations are represented. An r-ary relation $R^{\mathcal{A}}$ is represented by a sequence of $r \cdot |R^{\mathcal{A}}|$ integers consisting of all tuples contained in $R^{\mathcal{A}}$.

If we want the encoding of a structure to be unique, we may require that the universe and the relations are given in their natural (lexicographical) order. However, our algorithms are not based on this.

Example 2.1. The colored graph $\mathcal{G} \in \mathrm{STR}[\{P, Q, E\}]$ with $P = (1,1), Q = (2,1), E = (1,2)$ and $G = \{1,2,5,9\}$, $P^{\mathcal{G}} = \{2\}$, $Q^{\mathcal{G}} = \{1,5,9\}$, and $E^{\mathcal{G}} = \{(1,2),(2,1),(2,9),(9,2),(5,9),(9,5)\}$ (displayed in Figure 1) is encoded by the sequence

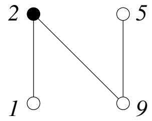

Figure 1.

$$[\mathcal{G}] := \big(3,1,1,2,1,1,2,4,1,3,6,1,2,5,9,2,1,5,9,1,2,2,1,2,9,5,9,9,2,9,5\big).$$

The *size* of a structure $\mathcal{A} \in \mathrm{STR}[\tau]$ is

$$||\mathcal{A}|| := ||\tau|| + 1 + |\tau| + |A| + \sum_{\substack{R\in\tau\\ r\text{-ary}}} r \cdot |R^{\mathcal{A}}| = O\Big(|\tau| + |A| + \sum_{R\in\tau} ||R^{\mathcal{A}}||\Big).$$

Finally, let us fix a reasonable encoding of formulas (by sequences of natural numbers) and let $||\varphi||$ denote the length of the encoding of a formula φ.

Trees. A *tree* is a connected acyclic graph $\mathcal{T}$. We always fix an (arbitrary) *root* $r^{\mathcal{T}} \in T$ in a tree $\mathcal{T}$. Then we have a natural partial order $\leq^{\mathcal{T}}$ on T, which is defined by

$$t \leq^{\mathcal{T}} u \iff t \text{ appears on the (unique) path from } r^{\mathcal{T}} \text{ to } u.$$

We say that u is a *child* of t or t is the *parent* of u if $(t,u) \in E^{\mathcal{T}}$ and $t \leq^{\mathcal{T}} u$. For every $t \in T$ we let $\mathcal{T}_t := \langle\{u \mid t \leq^{\mathcal{T}} u\}\rangle^{\mathcal{T}}$ be the subtree rooted at t. A tree $\mathcal{T}$ is *binary* if its root $r^{\mathcal{T}}$ has valence 0 or 2 and all other vertices have valence 1 or 3.

3 Tree-decompositions

In this section it will be convenient to work with hypergraphs (as an abstraction of relational structures). A *hypergraph* $\mathcal{H}$ is a pair $(H, E^{\mathcal{H}})$ consisting of a non-empty set H of *vertices* and a set $E^{\mathcal{H}}$ of non-empty subsets of H called *hyperedges*.

A *tree-decomposition* of a hypergraph $\mathcal{H}$ is a pair $(\mathcal{T}, (H_t)_{t\in T})$, where $\mathcal{T}$ is a tree and $(H_t)_{t\in T}$ a family of subsets of H (called the *blocks* of the decomposition) such that

(1) For every $v \in H$, the set $\{t \in T \mid v \in H_t\}$ is non-empty and connected (i.e. a subtree).

(2) For every $e \in E^{\mathcal{H}}$ there is a $t \in T$ such that $e \subseteq H_t$.

The *width* of a tree-decomposition $(\mathcal{T}, (H_t)_{t\in T})$ is $\max\{|H_t| \mid t \in T\} - 1$. The *tree-width* $\mathrm{tw}(\mathcal{H})$ of $\mathcal{H}$ is the minimal width of a tree-decomposition of $\mathcal{H}$.

A *tree-decomposition of a τ-structure* $\mathcal{A}$ is a tree decomposition of the hypergraph

$$\Big(A, \big\{\{a_1, \ldots, a_r\} \mid \exists R \in \tau,\ R\ r\text{-ary}, (a_1, \ldots, a_r) \in R^{\mathcal{A}}\big\}\Big);$$

the *tree-width* of $\mathcal{A}$ is defined accordingly.

Let $(\mathcal{T}, (H_t)_{t\in T})$ be a tree-decomposition of a hypergraph $\mathcal{H}$. For every $v \in H$ we let $\mathrm{node}(v)$ be the minimal node $t \in T$ (with respect to $\leq^{\mathcal{T}}$) sucht that $v \in H_t$, and for every $e \in E^{\mathcal{H}}$ we let $\mathrm{node}(e)$ be the minimal node $t \in T$ such that $e \subseteq H_t$.

Lemma 3.1. *There is a linear time algorithm that, given a hypergraph $\mathcal{H}$ and a tree-decomposition $(\mathcal{T}, (H_t)_{t\in T})$ of $\mathcal{H}$, computes the family $(\mathrm{node}(x))_{x\in H\cup E^{\mathcal{H}}}$.*

Proof: Let $\mathcal{H}$ be a hypergraph and $(\mathcal{T}, (H_t)_{t\in T})$ a tree-decomposition of $\mathcal{H}$.

We first compute a bijection $\mu : T \to \{1, \dots, |T|\}$ such that for all $t, t' \in T$ we have $t \leq^{\mathcal{T}} t' \implies \mu(t) \leq \mu(t')$. Then for every vertex $v \in H$ we let $\lambda(v) := \min\{\mu(t) \mid v \in H_t\}$. Clearly, such a labeling λ can be computed in time linear in the size of the tree-decomposition.

We observe that for every $v \in H$ we have $\mathrm{node}(v) = \mu^{-1}(\lambda(v))$ and for every $e \in E^{\mathcal{H}}$ we have $\mathrm{node}(e) = \max\{\mu^{-1}(\lambda(v)) \mid v \in e\}$ and use this to compute the mapping node in linear time. □

A tree-decomposition $(\mathcal{T}, (H_t)_{t\in T})$ of a hypergraph $\mathcal{H}$ is *reduced* if for all $t, u \in T, t \neq u$, we have $H_t \not\subseteq H_u$.

Lemma 3.2 (Bodlaender [6]). *Let $(\mathcal{T}, (H_t)_{t\in T})$ be a reduced tree-decomposition of a hypergraph $\mathcal{H}$. Then $|T| \leq |H|$.*

Proof: The proof is by induction on $|H|$, using the fact that the block of every leaf in a reduced tree-decomposition contains a vertex not contained in any other block. □

The next lemma is implicit in [17], but is also easy to prove directly.

Lemma 3.3. *There is a linear time algorithm that computes, given a hypergraph $\mathcal{H}$ and a tree-decomposition $(\mathcal{T}, (H_t)_{t\in T})$ of $\mathcal{H}$, a reduced tree-decomposition $(\mathcal{T}', (H'_t)_{t\in T'})$ of $\mathcal{H}$ such that for all $t' \in T'$ there is a $t \in T$ with $H'_{t'} = H_t$, and for all $t \in T$ there is a $t' \in T'$ such that $H_t \subseteq H'_{t'}$.*

Proof: Let $\mathcal{H}$ be a hypergraph and $(\mathcal{T}, (H_t)_{t\in T})$ a tree-decomposition of $\mathcal{H}$.

We start by computing labelings $\mu : T \to \{1, \dots, |T|\}$ and $\lambda : H \to \{1, \dots, |T|\}$ as in the proof of Lemma 3.1.

Call a node $t \in T$ *maximal* if there is no $u \in T$ such that $H_t \subset H_u$, and for all u with $H_t = H_u$ we have $t \leq^{\mathcal{T}} u$. Observe that a node t is maximal if, and only if, for the parent s of t we have $H_t \not\subseteq H_s$ and for every child u of t we have $H_t \not\subset H_u$. If t is a child of s, we have $H_t \subseteq H_s$ if, and only if, $|H_t| = |\{a \in H_t \mid \lambda(a) < \mu(t)\}|$. If u is a child of t, we have $H_t \subset H_u$ if, and only if, $|H_t| < |H_u|$ and $|H_t| = |\{a \in H_u \mid \lambda(a) < \mu(u)\}|$. We use this to find all maximal nodes in linear time.

We now compute the tree-decomposition $(\mathcal{T}', (H'_t)_{t\in T'})$ of $\mathcal{H}$ defined as follows: We let T' be the set of all maximal nodes of T, and $H'_t := H_t$ for all $t \in T'$. We pick an arbitrary $r \in T'$ to be the root of $\mathcal{T}'$. Then we define the edge relation inductively; the children of a vertex $t \in T'$ are all $u \in T'$ whose children are not yet defined such that there is no $v \in T'$ on the $\mathcal{T}$-path from t to u.

Clearly, $(\mathcal{T}', (H'_t)_{t\in T'})$ is a reduced tree-decomposition of $\mathcal{H}$. □

Theorem 3.4 (Bodlaender [6]). *There is a polynomial $p(X)$ and an algorithm that, given a hypergraph $\mathcal{H}$, computes a tree-decomposition of $\mathcal{H}$ of width $w := \mathrm{tw}(\mathcal{H})$ in time $2^{p(w)}|H|$.*

Actually, Bodlaender proves his theorem only for graphs. To extend it to hypergraphs, with every hypergraph $\mathcal{H}$ we associate a graph $\mathcal{G}_{\mathcal{H}}$ with the same vertex set as H and an edge between two vertices if they occur together in some hyperedge of $\mathcal{H}$. Then every tree-decomposition of $\mathcal{G}_{\mathcal{H}}$ is also a tree-decomposition of $\mathcal{H}$ and vice versa. It is easy to see that $\mathcal{G}_{\mathcal{H}}$ can be computed from $\mathcal{H}$ in time $O(||\mathcal{G}_{\mathcal{H}}||)$. There seems to be a slight problem because $\mathcal{G}_{\mathcal{H}}$ can be larger than $\mathcal{H}$. More precisely, the size of $\mathcal{G}_{\mathcal{H}}$ can be quadratic in the size of $\mathcal{H}$. However, a graph $\mathcal{G}$ of tree-width w has at most $w \cdot |G|$ edges [6]. Thus the size of $\mathcal{G}_{\mathcal{H}}$ is $O(\mathrm{tw}(\mathcal{H}) \cdot |H|)$.

In some situations, in particular when we are dealing with hypergraphs of small size, it is better to use the algorithm mentioned in the following Theorem 3.5 to compute tree-decompositions. The additional advantage of this algorithm is that, as opposed to Bodlaender's algorithm, the hidden constants in the bound on the running time are practically feasible.

Theorem 3.5 (Arnborg, Corneil, and Proskurowski [4]). *There is an algorithm that, given a hypergraph $\mathcal{H}$, computes a tree-decomposition of $\mathcal{H}$ of width $w := \text{tw}(\mathcal{H})$ in time $O(|H|^{w+2})$.*

4 Tree-like structures

In this section we present algorithms for evaluating monadic-second order formulas. We first deal with trees (Subsection 4.1) and then show how to extend the results to arbitrary structures parameterized by tree-width (Subsection 4.2). Our automata theoretic approach is based on ideas of Arnborg, Lagergren, and Seese [5].

4.1. Trees. For a finite alphabet Γ we let τ_Γ be the vocabulary consisting of a binary relation symbol E and a unary relation symbol P_γ for all $\gamma \in \Gamma$. A Γ-*tree* is a colored graph of vocabulary τ_Γ whose underlying graph is a binary tree, and a *colored tree* is a Γ-tree for some Γ.

We say that a vertex t of a colored tree $\mathcal{T}$ has *color* γ (and write $\gamma(t) := \gamma$), if $t \in P_\gamma^{\mathcal{T}}$.

A (bottom-up) Γ-*tree automaton* is a tuple $\mathfrak{A} = (Q, \delta, \Delta, F)$, where Q is a finite set, the set of *states*, $\Delta : \Gamma \to Q$ is the *starting function* , $F \subseteq Q$ is the set of *accepting states* and $\delta : [Q]^{\leq 2} \times \Gamma \to Q$ is the *transition function* (hence we only consider deterministic automata). Here, $[Q]^{\leq 2} := \{\{q, q'\} \mid q, q' \in Q\}$ is the set of singletons and pairs of elements of Q.

The *run* $\rho : T \to Q$ of $\mathfrak{A}$ on Γ-tree $\mathcal{T}$ is defined in a bottom-up manner (i.e., from leaves to the root): If t is a leaf, then $\rho(t) := \Delta(\gamma(t))$; if t has children s_1, s_2, then $\rho(t) := \delta(\{\rho(s_1), \rho(s_2)\}, \gamma(t))$. The automaton $\mathfrak{A}$ *accepts* $\mathcal{T}$ if $\rho(r^T) \in F$.

A class of colored trees is *recognizable*, if it is the class of colored trees accepted by some tree automaton.

Theorem 4.1 (Thatcher and Wright [18]). *Let Γ be a finite alphabet. A class of Γ-trees is recognizable if, and only if, it is definable by an* MSO$[\tau_\Gamma]$*-sentence.*

Furthermore, there is an algorithm that computes the tree automaton corresponding to a given MSO-*sentence.*

We use this theorem to efficiently evaluate the assignments satisfying a given MSO-formula. When computing $\varphi(\mathcal{A})$ for an MSO-formula $\varphi(X_1, \ldots, X_l, x_1, \ldots, x_m)$ and a structure $\mathcal{A}$, we have to handle sets $S \subseteq \text{Pow}(A)^l \times A^m$ for a set A and $l, m \geq 0$. While the appropriate data-structures to handle such S will be explained later, when describing the algorithms, let us define the *size* of S as

$$||S|| := \sum_{(B_1,\ldots,B_l,a_1,\ldots,a_m)\in S} \Big(\sum_{i=1}^{l} |B_i| + m\Big).$$

$||S||$ obviously is the size of some reasonable encoding of S.

Theorem 4.2. *There exists a function $f : \mathbb{N} \to \mathbb{N}$ and an algorithm that solves the evaluation problem for* MSO-*formulas on colored trees in time*

$$f(||\varphi||) \cdot \big(|T| + ||\varphi(\mathcal{T})||\big).$$

Proof: Without loss of generality we can restrict our attention to MSO-formulas without free first-order variables. Indeed, every MSO-formula $\varphi(X_1, \ldots, X_l, x_1, \ldots, x_m)$ can easily be translated to a formula $\varphi'(X_1, \ldots, X_l, Y_1, \ldots, Y_m)$ such that for every structure $\mathcal{A}$ and $\bar{B}, \bar{C} \subseteq A$

$$\mathcal{A} \models \varphi'(\bar{B}, \bar{C}) \iff \text{there are } c_1, \ldots, c_m \in A \text{ with } C_1 = \{c_1\}, \ldots, C_m = \{c_m\} \text{ and } \mathcal{A} \models \varphi(\bar{B}, \bar{c}).$$

Let Γ be an alphabet and $\varphi(X_1, \dots, X_k) \in \mathrm{MSO}[\tau_\Gamma]$. Theorem 4.1 only applies to sentences. Therefore, we replace the variables $X_1, \dots, X_k$ by new unary relation symbols appropriately. We set $\Gamma' := \Gamma \times \{0,1\}^k$. A Γ-tree $\mathcal{T}$ together with $B_1, \dots, B_k \subseteq T$ leads to a Γ'-tree $\mathcal{T}' := (\mathcal{T}; B_1, \dots, B_k)$ in a natural way: Denoting the color of $t \in T$ in $\mathcal{T}'$ by $\gamma'(t)$, we let

$$\gamma'(t) = (\gamma, \bar{\epsilon}) \quad \iff \quad \gamma(t) = \gamma \text{ and } (t \in B_i \Leftrightarrow \epsilon_i = 1) \text{ for } i = 1, \dots, k.$$

We call $\bar{\epsilon}$ the *additional color* of t.

The class of Γ'-trees

$$\{(\mathcal{T}; \bar{B}) \mid \mathcal{T} \text{ colored } \tau\text{-tree}, \bar{B} \subseteq T, \bar{B} \in \varphi(\mathcal{T})\}$$

is definable in MSO; hence, there is a Γ'-automaton $\mathfrak{A} = (Q, \Delta, \delta, F)$ recognizing it.

Let us fix a Γ-tree $\mathcal{T}$. We describe how to compute the set

$$\varphi(\mathcal{T}) = \{\bar{B} \subseteq T \mid \mathfrak{A} \text{ accepts } (\mathcal{T}; \bar{B})\}.$$

To do so, we pass through the tree $\mathcal{T}$ three times:

(1) Bottom-up. By induction from the leaves to the root, we first compute, for every $t \in T$, a set P_t of "potential states" at t: If t is a leaf, then $P_t := \{\Delta((\gamma(t), \bar{\epsilon})) \mid \bar{\epsilon} \in \{0,1\}^k\}$. For an inner vertex t with children s_1 and s_2, we set

$$P_t := \{\delta(\{q_1, q_2\}, (\gamma(t), \bar{\epsilon})) \mid q_1 \in P_{s_1},\, q_2 \in P_{s_2},\, \bar{\epsilon} \in \{0,1\}^k\}.$$

Then for all $t \in T$ and $q \in Q$ we have $q \in P_t$ if, and only if, there are sets $B_1, \dots, B_k \subseteq T$ such that for the run ρ of $\mathfrak{A}$ on $(\mathcal{T}; \bar{B})$ we have $\rho(t) = q$.

Note that if $P_r \cap F = \emptyset$ we have $\varphi(\mathcal{T}) = \emptyset$, and no further action is required.

(2) Top-down. Starting at the root $r := r^{\mathcal{T}}$ we compute, for every $t \in T$, the subset S_t of P_t of "success states" at t: We let $S_r := F \cap P_r$. If t has parent s and sibling t', then

$$S_t := \{q \in P_t \mid \text{there are } q' \in P_{t'}, \bar{\epsilon} \in \{0,1\}^k \text{ such that } \delta(\{q, q'\}, (\gamma(s), \bar{\epsilon})) \in S_s\}.$$

Then for all $t \in T$ and $q \in Q$ we have $q \in S_t$ if, and only if, there are sets $B_1, \dots, B_k \subseteq T$ such that $\mathfrak{A}$ accepts $(\mathcal{T}; \bar{B})$ and for the run ρ of $\mathfrak{A}$ on $(\mathcal{T}; \bar{B})$ we have $\rho(t) = q$.

(3) Bottom-up again. Recall that for $t \in T$, by $\mathcal{T}_t$ we denote the subtree of $\mathcal{T}$ rooted in t. For $t \in T$ and $q \in S_t$ we let

$$\mathrm{Sat}_{t,q} := \Big\{\bar{B} \subseteq T_t \;\Big|\; \begin{array}{l} \text{There are sets } B'_1, \dots, B'_k \subseteq T \text{ such that} \\ B'_i \cap T_t = B_i \text{ for } 1 \le i \le k, \\ \mathfrak{A} \text{ accepts } (\mathcal{T}; \bar{B}'), \\ \text{and for the run } \rho \text{ of } \mathfrak{A} \text{ on } (\mathcal{T}; \bar{B}') \text{ we have } \rho(t) = q \Big\}. \end{array}$$

We compute the sets $\mathrm{Sat}_{t,q}$ inductively from the leaves to the root. Let $t \in T$ and $q \in S_t$. Set $B^t_1 := \{t\}$ and $B^t_0 = \emptyset$. If t is a leaf, then

$$\mathrm{Sat}_{t,q} = \{(B^t_{\epsilon_1}, \dots, B^t_{\epsilon_k}) \mid \Delta((\gamma(t), \bar{\epsilon})) = q\}.$$

If t has children s and s', then

$$\begin{aligned} \mathrm{Sat}_{t,q} = \Big\{ (B_1 \cup B'_1 \cup B^t_{\epsilon_1}, \dots, B_k \cup B'_k \cup B^t_{\epsilon_k}) \;\Big|\; & \\ \bar{\epsilon} \in \{0,1\}^k, \text{there exist } q' \in S_s, q'' \in S_{s'} \text{such that} & \\ \bar{B} \in \mathrm{Sat}_{s,q'}, \bar{B}' \in \mathrm{Sat}_{s',q''}, \delta(\{q', q''\}, (\gamma(t), \bar{\epsilon})) = q \Big\}. & \end{aligned}$$

Note that $\varphi(\mathcal{T}) = \bigcup_{q \in S_r} \mathrm{Sat}_{r,q}$.

An algorithm evaluating a given formula on a given tree can now be described as follows:

Input: Colored tree $\mathcal{T}$, MSO-formula $\varphi(X_1, \dots, X_k)$.

1. Check if there is an alphabet Γ such that $\mathcal{T}$ is a colored Γ-tree and φ is an MSO$[\tau_\Gamma]$-formula; if this is not the case then return $\emptyset$.

2. Compute the Γ'-tree automaton $\mathfrak{A}$ corresponding to φ.

3. For all $t \in T$, compute P_t.

4. For all $t \in T$, compute S_t.

5. For all $t \in T$ and $q \in S_t$, compute $\text{Sat}_{t,q}$.

6. Return $\bigcup_{q \in S_r} \text{Sat}_{r,q}$.

Let us analyze the time required by this algorithm. Step 1 needs time linear in the size of the input. The time needed by Step 2 only depends on φ and thus is bounded by $f_1(||\varphi||)$ for a suitable function $f_1 : \mathbb{N} \to \mathbb{N}$. Note also that the size $||\mathfrak{A}||$ of the automaton $\mathfrak{A}$ only depends on φ. Steps 3 and 4 require one pass of the tree each, with an amount of work at each node that is linear in $||\mathfrak{A}||$. Thus the time needed for these two steps is $O(||\mathfrak{A}|| \cdot |T|)$, and the overall time needed in Steps 1–4 is $f_2(||\varphi||) \cdot |\mathcal{T}|$, for a suitable $f_2 : \mathbb{N} \to \mathbb{N}$.

Step 5, the third pass of the tree, is critical. We shall use the following two facts about the sets $\text{Sat}_{t,q}$.

Fact 1. $\text{Sat}_{t,q}$ is non-empty for all $t \in T, q \in S_t$.

This follows immediately from the definition of S_t.

Fact 2. $\text{Sat}_{t,q} \cap \text{Sat}_{t,q'} = \emptyset$ for all $t \in T$, $q, q' \in S_t$ such that $q \neq q'$.

This is due to the fact that we are dealing with a deterministic automaton. It is proved by a straightforward induction on t.

Remember that the sets $\text{Sat}_{t,q}$ are sets of k-tuples of sets. We use the following data structure to represent them: A subset of T is represented by a linked list with an additional pointer to the last element (cf. Figure 2). This enables us to form the union of two disjoint sets in constant time. A k-tuple of sets

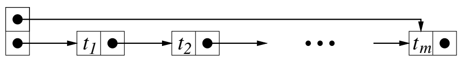

Figure 2. Representation of a set $B = \{t_1, \dots, t_m\} \subseteq T$.

is represented by an array of such lists, and a set of k-tuples of sets is represented as a linked list of such arrays, again with an additional pointer to the last element (cf. Figure 3).

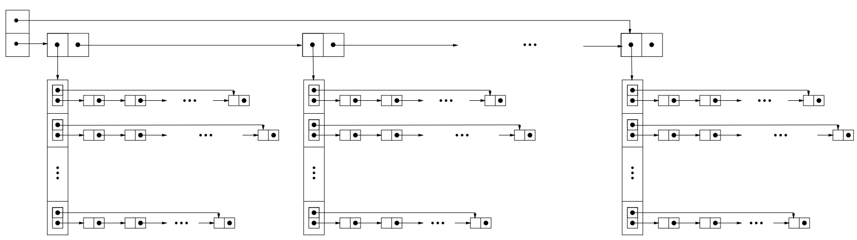

Figure 3. Representation of a set $\text{Sat}_{t,q}$.

Suppose now T', T'' are disjoint sets and S' and S'' are two non-empty sets of k-tuples of subsets of T' and T'', respectively. We let

$$\text{MERGE}(S', S'') := \{(B_1' \cup B_1'', \dots, B_k' \cup B_k'') \mid \bar{B}' \in S', \bar{B}'' \in S''\}.$$

If we are given S', S'' in the representation described above, then MERGE(S', S'') can be computed in time

$$O\Big(1 + k \cdot \big(||\text{MERGE}(S', S'')|| - ||S'|| - ||S''||\big)\Big), \tag{1}$$

where, as usual, for a set S of k-tuples of sets we let $||S|| = \sum_{\bar{B} \in S} \sum_{i=1}^{k} |B_i|$. Note that for S', S'' as above, the term $\big(||\text{MERGE}(S', S'')|| - ||S'|| - ||S''||\big)$ is always non-negative. To achieve the time bound (1), we use the actual representations of elements of S' and S'' in memory whenever this is possible and only copy them if they are used more than once.

Let us now analyze the time required to compute the sets $\text{Sat}_{t,q}$. Note that the number k and, for every $t \in T$, the size of the set S_t is bounded by $||\mathfrak{A}||$, the size of the automaton. Thus for every leaf $t \in T$ the sets $\text{Sat}_{t,q}$ can be computed in time $g_1(||\mathfrak{A}||)$ for a suitable function g_1.

For inner nodes t and a states $q \in S_t$, our goal is to compute $\text{Sat}_{t,q}$ in time $g_2(||\mathfrak{A}||) \cdot (|T_t| + ||\text{Sat}_{t,q}||)$ for a suitable function g_2. This yields the desired time bound.

Let t be an inner node with children s, s' and suppose we have already computed the sets $\text{Sat}_{s,q'}$ and $\text{Sat}_{s',q''}$ for all $q' \in S_s$, $q'' \in S_{s'}$. We assume that these sets are represented as described above. Then we can compute $\text{Sat}_{t,q}$ as follows:

$\text{Sat}_{t,q} := \emptyset$
for all $q' \in S_s, q'' \in S_{s'}$ **do**
    $N := \emptyset$
    for all $\bar{\epsilon} \in \{0,1\}^k$ **do**
        if $\delta(q', q'', (\gamma(t), \bar{\epsilon})) = q$ **then**
            $N := N \cup \{\bar{B}_{\bar{\epsilon}}\}$
        fi
    od
    if $N \neq \emptyset$ **then**
        $\text{Sat}_{t,q} := \text{MERGE}\big(\text{MERGE}(N, \text{Sat}_{s,q'}), \text{Sat}_{s',q''}\big)$
    fi
od

It is easy to see that this procedure works correctly; using (1), Fact 1, and Fact 2 we get the desired time bound. □

Remark 4.3. The function f in the statement of Theorem 4.2 is essentially determined by the size of the automaton to which the input formula is translated. It is well-known [16] that there is no elementary bound on the size of the automaton in terms of the size of the formula. The main factor contributing to the large automaton-size is the number of quantifier alternations in the formula; roughly there is one iterated exponential per quantifier alternation.

However, practical experience with such translations seems to indicate that very often the automaton does not get too big.

Corollary 4.4. *There exists a function $f : \mathbb{N} \to \mathbb{N}$ and an algorithm that solves the evaluation problem for* MSO*-formulas with no free set variables on colored trees in time*

$$f(||\varphi||) \cdot \big(|T| + |\varphi(\mathcal{T})|\big).$$

Proof: This follows from the fact that for a formula $\varphi(x_1, \dots, x_k)$ with no free set variables we have $||\varphi(\mathcal{T})|| = k|\varphi(\mathcal{T})| \leq ||\varphi|| \cdot |\varphi(\mathcal{T})|$. □

The following corollary easily follows from the proof of Theorem 4.2:

Corollary 4.5. *There exists a function $f : \mathbb{N} \to \mathbb{N}$ and an algorithm that, given a colored tree $\mathcal{T}$ and an MSO-formula $\varphi(X_1, \dots, X_l, x_1, \dots, x_m)$, computes sets $B_1, \dots, B_l \subseteq T$ and elements $a_1, \dots, a_m \in T$ such that $\mathcal{T} \models \varphi(B_1, \dots, B_l, a_1, \dots, a_m)$ in time*

$$f(||\varphi||) \cdot |T|.$$

4.2. Structures of bounded tree-width. It will be convenient for us to work with tree-decompositions of the following form: A *special tree-decomposition* of width w of a structure $\mathcal{A}$ is an ordered pair $(\mathcal{T}, (\bar{a}^t)_{t \in T})$ where $\mathcal{T}$ is a binary tree, every $\bar{a}^t := (a_0^t, \dots, a_w^t)$ is a $(w+1)$-tuple of elements of A, and $(\mathcal{T}, (\{a_0^t, \dots, a_w^t\})_{t \in T})$ is a tree-decomposition of $\mathcal{A}$ of width w. Let us mention explicitly that two distinct nodes of the tree of a special tree-decomposition may have identical blocks. It is easy to see that a given tree-decomposition of a structure can be transferred to a special tree-decomposition of the same width in linear time.

Let $\tau = \{R_1, \dots, R_m\}$, where R_i is r_i-ary for $1 \le i \le m$. With every special tree-decomposition $(\mathcal{T}, (\bar{a}^t)_{t \in T})$ of width w of a τ-structure $\mathcal{A}$ we associate a colored tree $\mathcal{T}^*$ with underlying tree $\mathcal{T}$ and coloring $\gamma(t) := (\gamma_1(t), \dots, \gamma_{m+2}(t))$ for $t \in T$, where

- $\gamma_1(t) := \{(i,j) \mid a_i^t = a_j^t\}$,
- $\gamma_2(t) := \begin{cases} \{(i,j) \mid a_i^t = a_j^s\} & \text{for the parent } s \text{ of } t \text{ if } t \neq r^{\mathcal{T}}, \\ \emptyset & \text{if } t = r^{\mathcal{T}}, \end{cases}$
- $\gamma_{i+2}(t) = \{(j_1, \dots, j_{r_i}) \mid (a_{j_1}^t, \dots, a_{j_{r_i}}^t) \in R_i^{\mathcal{A}}\}$ for $1 \le i \le m$.

The alphabet of $\mathcal{T}$ is

$$\begin{aligned} \Gamma(\tau, w) := \quad & \mathrm{Pow}(\{0, \dots, w\}^2) \times \mathrm{Pow}(\{0, \dots, w\}^2) \\ & \times \mathrm{Pow}(\{0, \dots, w\}^{r_1}) \times \cdots \times \mathrm{Pow}(\{0, \dots, w\}^{r_m}). \end{aligned}$$

We observe the following:

Lemma 4.6. *Given a τ-structure $\mathcal{A}$ and a special tree-decomposition $(\mathcal{T}, (\bar{a}^t)_{t \in T})$ of width w of $\mathcal{A}$, the corresponding $\Gamma(\tau, w)$-tree $\mathcal{T}^*$ can be computed in time*

$$f(\tau, w)|T|$$

for a suitable function f.

Recall that for an element $a \in A$ we let $\mathrm{node}(a)$ be the minmal $t \in T$ with respect to $\le^{\mathcal{T}}$ such that $a \in \{a_0^t, \dots, a_w^t\}$. Define $\bar{U}(a) := (U_0(a), \dots, U_w(a))$ by

$$U_i(a) := \begin{cases} \{\mathrm{node}(a)\} & \text{if } a_i^{\mathrm{node}(a)} = a \text{ and } a_j^{\mathrm{node}(a)} \neq a \text{ for } 1 \le j < i, \\ \emptyset & \text{otherwise,} \end{cases}$$

for $0 \le i \le w$. For a subset $B \subseteq A$ we let $U_i(B) := \bigcup_{a \in B} U_i(a)$ and $\bar{U}(B) := (U_0(B), \dots, U_w(B))$. Note that for subsets $U_0, \dots, U_w \subseteq T$ there exists an $a \in A$ such that $\bar{U} = \bar{U}(a)$ if, and only if,

(1) $\bigcup_{i=0}^{w} U_i$ is a singleton.

(2) For all $t \in T$ and $0 \le i < j \le w$: If $t \in U_j$ then $(i,j) \notin \gamma_1(t)$.

(3) For all $t \in T$, $t \neq r^{\mathcal{T}}$, with parent s and $0 \le i, j \le w$: If $t \in U_i$ then $(i,j) \notin \gamma_2(t)$.

Moreover, there is a subset $B \subseteq A$ with $\bar{U} = \bar{U}(B)$ just in case the conditions (2) and (3) are fulfilled.

We can pass from B to $\bar{U}(B)$ and also from $\bar{U}(B)$ to B in linear time (note that $B = \dot{\bigcup}_{0 \le i \le w} \{a_i^t \mid t \in U_i(B)\}$).

By the characterizations just given it is straightforward to exhibit MSO-formulas $\mathrm{Elem}(X_0,\ldots,X_w)$ and $\mathrm{Set}(X_0,\ldots,X_w)$ such that for arbitrary $U_0,\ldots,U_w \subseteq T$,

$$\begin{aligned} \mathcal{T}^* \models \mathrm{Elem}(\bar{U}) \quad &\Longleftrightarrow \quad \text{there is an } a \in A \text{ with } \bar{U} = \bar{U}(a); \\ \mathcal{T}^* \models \mathrm{Set}(\bar{U}) \quad &\Longleftrightarrow \quad \text{there is a subset } B \subseteq A \text{ with } \bar{U} = \bar{U}(B). \end{aligned}$$

This observation shows how to guarantee the second statement of the following lemma.

Lemma 4.7. *Every* MSO*-formula* $\varphi(X_1,\ldots,X_k,y_1,\ldots,y_l)$ *can be effectively translated to a formula* $\varphi^*(\bar{X}_1,\ldots,\bar{X}_k,\bar{Y}_1,\ldots,\bar{Y}_l)$ *such that:*

(1) For all $B_1,\ldots,B_k \subseteq A, a_1,\ldots,a_l \in A$ *we have*

$$\mathcal{A} \models \varphi(B_1,\ldots,B_k,a_1,\ldots,a_l) \iff \mathcal{T}^* \models \varphi^*(\bar{U}(B_1),\ldots,\bar{U}(B_k),\bar{U}(a_1),\ldots,\bar{U}(a_l)).$$

(2) For all $\bar{U}_1,\ldots,\bar{U}_k,\bar{V}_1,\ldots,\bar{V}_l \subseteq T$ *such that* $\mathcal{T}^* \models \varphi^*(\bar{U}_1,\ldots,\bar{U}_k,\bar{V}_1,\ldots,\bar{V}_l)$ *there exist* $B_1,\ldots,B_k \subseteq A$*,* $a_1,\ldots,a_l \in A$ *such that* $\bar{U}_i = \bar{U}(B_i)$ *for* $1 \leq i \leq k$ *and* $\bar{V}_i = \bar{U}(a_i)$ *for* $1 \leq i \leq l$*.*

Proof: The proof is by induction on φ, the case of atomic φ being the only nontrivial part. For this purpose we say that $\bar{V} = (V_0,\ldots,V_w)$ with $V_0,\ldots,V_w \subseteq T$ is *closed*, if the following hold:

– If $t \in V_i$ and $(i,j) \in \gamma_1(t)$ then $t \in V_j$.

– If t has parent s, $s \in V_j$ and $(i,j) \in \gamma_2(t)$ then $t \in V_i$.

Clearly, there is an MSO-formula $\mathrm{Closed}(X_0,\ldots,X_w)$ such that

$$\mathcal{T}^* \models \mathrm{Closed}(\bar{V}) \quad \Longleftrightarrow \quad \bar{V} \text{ is closed.}$$

Note that, for $a \in A$ and closed $\bar{V}$, if $\bar{U}(a) \subseteq \bar{V}$, $t \in T$, $0 \leq i \leq w$ and $a_i^t = a$ then $t \in V_i$. Now it is easy to define the translation $\varphi \mapsto \varphi^*$ required in the lemma, e.g. for $\varphi = Ry_1\ldots y_r$, where $R = R_i$ is $r = r_i$-ary, we set

$$\begin{aligned} \varphi^*(\bar{Y}_1,\ldots,\bar{Y}_r) \quad := \quad & \mathrm{Elem}(\bar{Y}_1) \wedge \ldots \wedge \mathrm{Elem}(\bar{Y}_r) \wedge \\ & \forall \bar{Z}_1 \ldots \forall \bar{Z}_r \Big(\bigwedge_{i=1}^{r} \big(\bar{Y}_i \subseteq \bar{Z}_i \wedge \mathrm{Closed}(\bar{Z}_i)\big) \rightarrow \\ & \qquad \exists x \big(\bigvee_{i_1,\ldots,i_r=0}^{w} (Z_{1i_1}x \wedge \ldots \wedge Z_{ri_r}x \wedge \bigvee_{\substack{\gamma \in \Gamma(\tau,w) \\ (i_1,\ldots,i_r) \in \gamma_{i+2}}} P_\gamma x)\big)\Big). \end{aligned}$$

Theorem 4.8. *There exists a function* $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ *and an algorithm that solves the evaluation problem for* MSO*-formulas in time*

$$f\big(||\varphi||, \mathrm{tw}(\mathcal{A})\big) \cdot \Big(|A| + ||\varphi(\mathcal{A})||\Big).$$

Proof: The algorithm proceeds as follows:

Input: Structure $\mathcal{A}$, MSO-formula φ

1. Check if $\mathcal{A}$ and φ have the same vocabulary, say τ; if this is not the case then return $\emptyset$.
2. Compute a special tree-decomposition $(\mathcal{T}, (\bar{a}^t)_{t\in T})$ of tree-width $w := \mathrm{tw}(\mathcal{A})$ of $\mathcal{A}$.
3. Compute the corresponding $\Gamma(\tau,w)$-tree $\mathcal{T}^*$.
4. Compute the formula φ^*.
5. Compute $\varphi^*(\mathcal{T}^*)$.

6. Compute $\varphi(\mathcal{A})$.

Line 1 requires time linear in the size of the input. Without loss of generality we can assume that $||\tau|| \leq ||\varphi||$. Line 2 requires time $f_1(||\tau||, w)|A|$ for a suitable function f_1 (by Theorem 3.4). Line 3 requires time $f_2(||\tau||, w)|A|$ for a suitable function f_2 (by Lemma 4.6).

Line 4 requires time $f_3(||\varphi||)$ (by Lemma 4.7). Certainly, the length of φ^* is recursively bounded in terms of $||\varphi||$.

Line 5 require time $f_4(||\varphi^*||) \cdot (|T| + ||\varphi^*(\mathcal{T}^*)||)$ (by Theorem 4.2). Finally, given $\varphi^*(\mathcal{T}^*)$ and the mapping $\bar{U}(a) \mapsto a$, the output $\varphi(\mathcal{A})$ can be computed in linear time.

Putting everything together, we get the desired time bound. □

Corollary 4.9. *There exists a function $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ and an algorithm that solves the evaluation problem for* MSO*-formulas with no free set variables in time*

$$f\big(||\varphi||, \mathrm{tw}(\mathcal{A})\big) \cdot \Big(|A| + |\varphi(\mathcal{A})|\Big).$$

Corollary 4.10. *There exists a function $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ and an algorithm that, given a structure $\mathcal{A}$ and an MSO-formula $\varphi(X_1, \ldots, X_l, x_1, \ldots, x_m)$, computes sets $B_1, \ldots, B_l \subseteq A$ and elements $a_1, \ldots, a_m \in A$ such that $\mathcal{A} \models \varphi(B_1, \ldots, B_l, a_1, \ldots, a_m)$ in time*

$$f(||\varphi||, \mathrm{tw}(\mathcal{A})) \cdot |A|.$$

Remark 4.11. There is a trick that can sometimes improve the running time of our algorithms considerably, in particular if the arity of the vocabulary is high.

For every vocabulary τ we let τ_b be the vocabulary that contains a unary relation symbol P_R for every $R \in \tau$ and binary relation symbols $E_1, \ldots, E_s$, where s is the arity of τ. Then with every τ-structure $\mathcal{A}$ we associate a τ_b-structure $\mathcal{A}_b$, the *bipartite structure* associated with $\mathcal{A}$. The universe consists of A together with a new vertex $b_{R\bar{a}}$ for all $R \in \tau$ and $\bar{a} \in R^{\mathcal{A}}$. The relation E_i holds for all pairs $(a_i, b_{Ra_1 \ldots a_r})$.

Then $\mathcal{A}_b$ can be computed from $\mathcal{A}$ in linear time, and we have $|A_b| = O(||\mathcal{A}||)$. Moreover, we have $\mathrm{tw}(\mathcal{A}_b) \leq \mathrm{tw}(\mathcal{A}) + 1$. If the arity of τ is at most $\mathrm{tw}(\mathcal{A})$, this can be improved to $\mathrm{tw}(\mathcal{A}_b) \leq \mathrm{tw}(\mathcal{A})$. The tree-width of $\mathcal{A}_b$ can be considerably smaller than that of $\mathcal{A}$. For example, if R is a 1000-ary relation, then the tree width of the structure $\mathcal{A} = (\{1, \ldots, 1000\}, \{(1, \ldots, 1000)\})$ is 999, whereas the tree-width of $\mathcal{A}_b$ is 1. Let us remark that for graphs $\mathcal{G}$ we have $\mathrm{tw}(\mathcal{G}) = \mathrm{tw}(\mathcal{G}_b)$.

It is easy to see that there is a linear-time algorithm that associates with every MSO-formula φ an MSO-formula φ_b such that for all structures $\mathcal{A}$ we have $\varphi(\mathcal{A}) = \varphi_b(\mathcal{A}_b)$.

Thus to evaluate MSO-formulas we can also proceed as follows: Given a formula φ and a structure $\mathcal{A}$, we first compute φ_b and $\mathcal{A}_b$. Then we compute $\varphi_b(\mathcal{A}_b)$ using our algorithms. By Theorem 4.8, this requires time

$$O\Big(f\big(||\varphi_b||, \mathrm{tw}(\mathcal{A}_b)\big) \cdot \big(||\mathcal{A}|| + ||\varphi(\mathcal{A})||\big)\Big),$$

which can be much better than $f(||\varphi||, \mathrm{tw}(\mathcal{A})) \cdot \big(|A| + ||\varphi(\mathcal{A})||\Big)$.

There is another advantage in working with the structure $\mathcal{A}_b$ instead of $\mathcal{A}$: MSO becomes more expressive. Intuitively, the reason is that in $\mathcal{A}_b$ we can talk about sets of "edges". For example, it is easy to see that there is an MSO-formula $\varphi(X)$ such that for every graph $\mathcal{G}$, $\varphi(\mathcal{G}_b)$ consists of all sets $\{b_{Eab} \mid ab \in H\}$, where H ranges over the edge sets of all Hamiltonian cycles of $\mathcal{G}$. Thus in a sense φ defines the set of all Hamiltonian cycles of a graph. Clearly, this is not possible by an MSO-formula in the original graph $\mathcal{G}$. As a matter of fact, there is not even an MSO-sentence that holds in a graph $\mathcal{G}$ if, and only if, $\mathcal{G}$ is Hamiltonian [14].

5 Tree-like formulas

In this section, we restrict our attention to first-order formulas. Recall that *atomic formulas*, or *atoms*, are formulas of the form $x = y$ or $Rx_1 \ldots x_r$ for an r-ary relation symbol R. The set of all atoms occurring in a formula φ is denoted by $\text{at}(\varphi)$. *Literals* are atomic or negated atomic formulas. The set of all variables occurring in a formula φ is denoted by $\text{var}(\varphi)$, the set of free variables of φ by $\text{free}(\varphi)$.

With every formula φ we associate a hypergraph

$$\mathcal{H}_\varphi := \big(\text{var}(\varphi), \{\text{var}(\alpha) \mid \alpha \in \text{at}(\varphi)\}\big).$$

Definition 5.1. A *tree-decomposition* of a formula φ is a tree-decomposition of $\mathcal{H}_\varphi$.

A tree-decomposition $(\mathcal{T}, (X_t)_{t\in T})$ of a formula φ is *strict* if there exists a $t \in T$ such that $\text{free}(\varphi) \subseteq X_t$.

For arbitrary first-order formulas, tree-decomposition are quite meaningless, as the following example shows:

Example 5.2. Let φ be a formula, and suppose that the vocabulary of φ is r-ary, for some $r \geq 2$. We show that φ is equivalent to a formula φ' that has a very simple tree-decomposition of width r.

Let $y_1, \dots, y_r \notin \text{var}(\varphi)$. To obtain φ', we replace every atomic subformula $\alpha(x_1, \dots, x_s)$ of φ by the formula $\exists y_1 \dots \exists y_s \big(x_1 = y_1 \wedge \dots \wedge x_s = y_s \wedge \alpha(y_1, \dots, y_s)\big)$. Then obviously, φ and φ' are equivalent.

Let $\mathcal{T}$ be the tree with root $r^\mathcal{T}$ and vertex t_x adjacent to $r^\mathcal{T}$ for every $x \in \text{var}(\varphi)$. Let $X_{r^\mathcal{T}} := \{y_1, \dots, y_r\}$ and $X_{t_x} := X_{r^\mathcal{T}} \cup \{x\}$ for $x \in \text{var}(\varphi)$. Then $(\mathcal{T}, (X_t)_{t\in T})$ is a tree-decomposition of φ'.

However, tree-decompositions turn out to be quite useful when it comes to evaluating formulas of a very simple form, which are known as *conjunctive queries*.

5.1. Acyclic Conjunctive Queries. A *conjunctive query* is a formula of the form $\exists y_1 \dots \exists y_m \bigwedge_{i=1}^n \alpha_i$ with atomic formulas $\alpha_1, \dots, \alpha_n$. In this subsection we explain an algorithm due to Yannakakis for evaluating *acyclic* conjunctive queries.

We have to recall some basic notions of relational database theory. An X-relation $\mathcal{R}$, for a finite set X, is a finite set of mappings with domain X. We let $\text{range}(\mathcal{R}) := \bigcup_{\gamma\in\mathcal{R}} \gamma(X)$. We think of an X-relation as an $|X|$-ary relation on $\text{range}(\mathcal{R})$ in which we have associated a name (an element of X) with every place of the relation. Usually, X is a set of variables and $\text{range}(\mathcal{R})$ is contained in the universe of some structure. For $Y \subseteq X$, the Y-projection of an X-relation $\mathcal{R}$ is the set $\pi_Y(\mathcal{R}) := \{\gamma|_Y \mid \gamma \in \mathcal{R}\}$. For sets X, Y of variables, the *join* of an X-relation $\mathcal{R}$ and a Y-relation $\mathcal{S}$ is the $X \cup Y$-relation

$$\mathcal{R} \bowtie \mathcal{S} := \{\gamma : X \cup Y \to A \mid \gamma|_X \in \mathcal{R}, \gamma|_Y \in \mathcal{S}\}.$$

For every formula $\varphi(x_1, \dots, x_l)$ and structure $\mathcal{A}$ the set $\varphi(\mathcal{A})$ is an $\{x_1, \dots, x_l\}$-relation over A. On the logical level, projections correspond to existential quantifications and joins correspond to conjunctions. In particular, for a conjunctive query $\varphi(x_1, \dots, x_l) = \exists y_1 \dots \exists y_m \bigwedge_{i=1}^n \alpha_i$ we have

$$\varphi(\mathcal{A}) = \pi_{\{x_1,\dots,x_l\}}\big(\alpha_1(\mathcal{A}) \bowtie \dots \bowtie \alpha_n(\mathcal{A})\big).$$

In an appendix, we explain how to represent X-relations on a RAM and how to compute projections and joins efficiently. The *size* $||\mathcal{R}||$ of an $|X|$-relation $\mathcal{R}$ is defined to be $2 + |X| + |X| \cdot |\mathcal{R}|$. By Theorem A.5, the projection $\pi_Y(\mathcal{R})$ of an X-relation $\mathcal{R}$ to a $Y \subseteq X$ can be computed in time $O(||\mathcal{R}||)$, and the join $\mathcal{R} \bowtie \mathcal{S}$ of an X-relation $\mathcal{R}$ and a Y-relation $\mathcal{S}$ can be computed in time $O\big(||\mathcal{R}|| + ||\mathcal{S}|| + ||\mathcal{R} \bowtie \mathcal{S}||\big)$.

The following two lemmas describe Yannakakis's basic algorithm. The idea behind these lemmas is the following: We want to evaluate a conjunctive query φ in a structure $\mathcal{A}$. Suppose we can efficiently compute a tree-decomposition $(\mathcal{T}, (X_t)_{t\in T})$ of φ and, for every $t \in T$, the X_t-relation

$$\mathcal{P}_t := \bigotimes_{\substack{\alpha\in\text{at}(\varphi)\\ \text{var}(\alpha)\subseteq X_t}} \alpha(\mathcal{A}).$$

Then, noting that $\varphi(\mathcal{A}) := \pi_{\text{free}(\varphi)}\big(\bowtie_{t\in T} \mathcal{P}_t\big)$, we can use Lemma 5.4 to compute $\varphi(\mathcal{A})$. Moreover, if the tree-decomposition is strict, i.e. if $\text{free}(\varphi) \subseteq X_t$ for some $t \in T$, we can even do better using Lemma 5.3.

Lemma 5.3. *There is an algorithm solving the following problem in time* $O\big(|T| \cdot \max_{t\in T}||\mathcal{P}_t||\big)$*:*

Input:	Tree-decomposition $(\mathcal{T}, (X_t)_{t\in T})$, an X_t-relation $\mathcal{P}_t$ for every $t \in T$.
Problem:	Compute $\mathcal{R}_t := \pi_{X_t}\big(\bowtie_{t\in T}\mathcal{P}_t\big)$ for all $t \in T$.

Proof: The algorithm passes the tree twice.

(1) Bottom-up. For every $t \in T$ we let $\mathcal{Q}_t := \pi_{X_t}\Big(\bowtie_{u \geq^{\mathcal{T}} t} \mathcal{P}_u\Big)$.

Then if t is a leaf, we have

$$\mathcal{Q}_t = \mathcal{P}_t, \tag{2}$$

and if t is an inner node with children $t_1, \dots, t_m$, then

$$\mathcal{Q}_t = \pi_{X_t}\Big(\mathcal{P}_t \bowtie \bowtie_{1\leq j\leq m} \mathcal{Q}_{t_j}\Big) = \mathcal{P}_t \bowtie \pi_{X_t}\Big(\bowtie_{1\leq j\leq m} \mathcal{Q}_{t_j}\Big) = \mathcal{P}_t \bowtie \bowtie_{1\leq j\leq m} \pi_{X_t}(\mathcal{Q}_{t_j}). \tag{3}$$

To establish the last equality we use the fact that for $1 \leq i < j \leq m$ we have $X_{t_i} \cap X_{t_j} \subseteq X_t$, because for every $x \in \text{var}(\varphi)$ the set $\{t \in T \mid x \in X_t\}$ is connected in $\mathcal{T}$.

Our algorithm uses (2) and (3) to inductively compute the sets $\mathcal{Q}_t$ for all $t \in T$. This amounts to computing at most one projection and one join of relations of size at most $\max_{t\in T}||\mathcal{P}_t||$ for every tree-node.

(2) Top-down. Note that for the root $r := r^{\mathcal{T}}$ we have $\mathcal{R}_r = \mathcal{Q}_r$. For a node $t \in T \setminus \{r\}$ with parent s we have

$$\mathcal{R}_t = \pi_{X_t}(\mathcal{R}_s) \bowtie \mathcal{Q}_t.$$

Thus we can compute $\mathcal{R}_t$ for all $t \in T$ in top down manner. Again this requires to compute at most one projection and one join per tree-node. Noting that $\mathcal{R}_t \subseteq \mathcal{Q}_t \subseteq \mathcal{P}_t$, we see that this can be done within the desired time bounds. □

Lemma 5.4. *There is an algorithm solving the following problem in time $O\big(|T|\cdot\max_{t\in T}||\mathcal{P}_t||\cdot(||\mathcal{S}||+1)\big)$:*

Input:	Tree-decomposition $(\mathcal{T}, (X_t)_{t\in T})$, a subset $X \subseteq \bigcup_{t\in T} X_t$, an X_t-relation $\mathcal{P}_t$ for every $t \in T$.
Problem:	Compute $\mathcal{S} := \pi_X\big(\bowtie_{t\in T}\mathcal{P}_t\big)$.

Proof: We first compute the family $(\mathcal{R}_t)_{t\in T}$ as in Lemma 5.3.

For every $t \in T$, let $Y_t := \bigcup_{u\geq^{\mathcal{T}} t}(X \cap X_u)$. If t has a parent s, then let $Z_t := X_t \cap X_s$, and let $Z_r := \emptyset$. Let

$$\mathcal{S}_t := \pi_{Y_t\cup Z_t}\big(\bowtie_{u\geq^{\mathcal{T}} t} \mathcal{R}_u\big).$$

Then $\mathcal{S} = \mathcal{S}_r$.

We can compute the relations $\mathcal{S}_t$ inductively in a bottom-up manner, noting that for a leaf t we have $\mathcal{S}_t = \pi_{Y_t\cup Z_t}(\mathcal{R}_t)$ and for an inner node t with children $t_1, \dots, t_m$ we have

$$\mathcal{S}_t = \pi_{Y_t\cup Z_t}\Big(\mathcal{R}_t \bowtie \mathcal{S}_{t_1} \bowtie \cdots \bowtie \mathcal{S}_{t_m}\Big).$$

Note that the size of all intermediate joins is bounded by $||\mathcal{R}_t|| \cdot ||\mathcal{S}_t|| \leq ||\mathcal{P}_t|| \cdot ||\mathcal{S}||$. □

Definition 5.5. A hypergraph $\mathcal{H}$ is *acyclic* if there is a tree-decomposition $(T, (H_t)_{t\in T})$ of $\mathcal{H}$ such that for every $t \in T$ there exists a hyperedge $e \in E^H$ such that $e = X_t$. We call such a tree-decomposition a *chordal decomposition* of $\mathcal{H}$.

A conjunctive query φ is *acyclic* if its hypergraph $\mathcal{H}_\varphi$ is. φ is *strictly acyclic* if $\mathcal{H}_\varphi$ has a chordal decomposition that is a strict tree-decomposition of φ.

Observe that every chordal decomposition of a strictly acyclic query is a strict tree-decomposition, because if φ is strictly acyclic, then there exists an atom $\alpha \in \text{at}(\varphi)$ such that $\text{free}(\varphi) \subseteq \text{var}(\alpha)$.

Theorem 5.6 (Tarjan, Yannakakis [17]). *Given a hypergraph $\mathcal{H}$, it can be decided in linear time if $\mathcal{H}$ is acyclic. If this is the case, a chordal decomposition of $\mathcal{H}$ can be computed in linear time.*

Let $(\mathcal{T}, (H_t)_{t\in T})$ be a reduced chordal decomposition of an acyclic hypergraph $\mathcal{H}$. Observe that there is a bijection between the vertices of $\mathcal{T}$ and the maximal (with respect to set inclusion) hyperedges of $\mathcal{H}$. Thus for every $t \in T$ there is exactly one $e \in E^H$ such that $\text{node}(e) = t$ and $e = X_t$ (where $\text{node}(e)$ is defined as in Lemma 3.1).

If $(\mathcal{T}, (H_t)_{t\in T})$ is a tree-decomposition of a formula φ and $\alpha := Rx_1 \dots x_r \in \text{at}(\varphi)$, then we let $\text{node}(\alpha) := \text{node}(\{x_1, \dots, x_r\})$. The proof of Lemma 3.1 shows that the family $(\text{node}(\alpha))_{\alpha\in\text{at}(\varphi)}$ can be computed in linear time.

Theorem 5.7 (Yannakakis [22]). *There is an algorithm that solves the evaluation problem for acyclic conjunctive queries in time $O\big(||\varphi|| \cdot ||\mathcal{A}|| \cdot ||\varphi(\mathcal{A})||\big)$.*

For strictly acyclic conjunctive queries, this can be improved to $O\big(||\varphi|| \cdot ||\mathcal{A}||\big)$.

Proof: Given $\mathcal{A}$ and φ, we first compute a chordal decomposition $(\mathcal{T}, (X_t)_{t\in T})$ of φ. By Theorem 5.6, this is possible in time $O(||\varphi||)$. By Lemma 3.3, we can actually assume that this decomposition is reduced. Then by Lemma 3.2 we have $|T| \le ||\varphi||$. We compute $\text{node}(\alpha)$ for all $\alpha \in \text{at}(\varphi)$.

Clearly, for every atomic formula $\alpha(\bar{x})$ we can compute $\alpha(\mathcal{A})$ in time $O(||\mathcal{A}||)$.

For every $t \in T$ we let

$$\mathcal{P}_t := \bowtie_{\substack{\alpha\in\text{at}(\varphi)\\ \text{node}(\alpha)=t}} \alpha(\mathcal{A}).$$

Then $\varphi(\mathcal{A}) = \pi_{\text{free}(\varphi)}\big(\bowtie_{t\in T} \mathcal{P}_t\big)$. For $t \in T$ we let $\alpha_t \in \text{at}(\varphi)$ such that $\text{var}(\alpha_t) = X_t$. Then $\mathcal{P}_t \subseteq \alpha_t(\mathcal{A})$, and thus $||\mathcal{P}_t|| \le ||\alpha_t(\mathcal{A})|| \le ||\mathcal{A}||$. Furthermore, we can compute the family $(\mathcal{P}_t)_{t\in T}$ in time $O(||\varphi|| \cdot ||\mathcal{A}||)$, because we only have to compute at most one join per atom of φ, and if these joins are computed in the right order, the intermediate relations obtained while computing $\mathcal{P}_t$ are all contained in $\alpha_t(\mathcal{A})$.

We can now apply Lemma 5.4. The stronger statement for strictly acyclic φ follows from Lemma 5.3. □

Remark 5.8. Actually, the number $||\mathcal{A}||$ in the statement of Theorem 5.7 can be replaced by $\text{rel-size}(\mathcal{A}) := \max\{||\mathcal{R}^{\mathcal{A}}|| \mid R \in \tau\}$, where τ is the vocabulary of $\mathcal{A}$.

Definition 5.9. The *tree-width* $\text{tw}(\varphi)$ of a formula φ is the maximal width of a tree-decomposition of φ. The *strict tree-width* $\text{stw}(\varphi)$ of φ is the maximal width of a strict tree-decomposition of φ.

Corollary 5.10 (Chekuri and Rajaraman [7]). *The evaluation problem for conjunctive queries can be solved in time $O\big(||\varphi||^{w+2} + ||\varphi|| \cdot (|A|^{w+1} + ||\mathcal{A}||) \cdot ||\varphi(\mathcal{A})||\big)$, where $w := \text{tw}(\varphi)$, and in time $O\big(||\varphi||^{s+2} + ||\varphi|| \cdot (|A|^{s+1} + ||\mathcal{A}||)\big)$, where $s := \text{stw}(\varphi)$.*

Proof: The proof is analogous to the proof of Theorem 5.7, using Theorem 3.5 instead of Theorem 5.6 to compute a tree-decomposition $(\mathcal{T}, (X_t)_{t\in T})$ of width w. The main difference is that the X_t-relations $\mathcal{P}_t$

are no longer contained in a relation of $\mathcal{A}$. But since here we have $|X_t| \leq w + 1$, we have $|\mathcal{P}_t| \leq |A|^{w+1}$ and thus $||\mathcal{P}_t|| = O(|A|^{w+1})$. The term $||\varphi|| \cdot ||\mathcal{A}||$ takes care of the evaluation of the atoms.

There a slight problem in the extension to strict tree-width, since it is not the case that any tree-decomposition of φ is also a strict tree-decomposition of φ. However, a strict tree-decomposition of φ is simply a tree-decomposition of the hypergraph obtained from $\mathcal{H}_\varphi$ adding a hyperedge free(φ). By Theorem 3.5, we can compute such a decomposition of minimal width s in time $O(||\varphi||^{s+2})$. □

Remark 5.11. Note that in most situations we will have $||\varphi|| \leq |A|$. Then the terms $||\varphi||^{w+2}$ and $||\varphi||^{s+2}$ can be omitted. If we use Bodlaender's [6] algorithm instead of Arnborg, Corneil, and Proskurowski's [4] to compute the tree-decompositions, we can replace these terms by $2^{p(w)}||\varphi||$ and $2^{p(s)}||\varphi||$, respectively (for a suitable polynomial p).

Remark 5.12. Recall Remark 4.11, where we evaluated a query by first translating the input structure $\mathcal{A}$ to a bipartite structure $\mathcal{A}_b$ and the input formula φ to a formula φ_b such that $\varphi(\mathcal{A}) = \varphi_b(\mathcal{A}_b)$. Then we evaluated φ_b on $\mathcal{A}_b$ instead.

If φ is a conjunctive query (and the translation is done in the right way), then φ_b will be a conjunctive query and $||\varphi_b|| = O(||\varphi||)$. The hypergraph of this conjunctive query φ_b is

$$\mathcal{B}_\varphi := \Big(\mathrm{var}(\varphi) \cup \mathrm{at}(\varphi), \big\{\{x, \alpha\} \bigm| x \in \mathrm{var}(\alpha)\big\}\Big),$$

which is a bipartite graph. As in Remark 4.11 we have $\mathrm{tw}(\mathcal{B}_\varphi) \leq \mathrm{tw}(\varphi) + 1$, and $\mathrm{tw}(\mathcal{B}_\varphi) \leq \mathrm{tw}(\varphi)$ if the arity of the vocabulary of φ is at most $\mathrm{tw}(\varphi)$.

Recalling that $|A_b| = O(||\mathcal{A}||)$, by Corollary 5.10, we can thus evaluate φ in time $O\big(||\varphi||^{b+2} + ||\varphi|| \cdot ||\mathcal{A}||^{b+1} \cdot ||\varphi(\mathcal{A})||\big)$, where $b := \mathrm{tw}(\mathcal{B}_\varphi)$.

Actually, Chekuri and Rajaraman [7] defined the tree-width of a conjunctive query to be $\mathrm{tw}(\mathcal{B}_\varphi)$ instead of $\mathrm{tw}(\mathcal{H}_\varphi)$.

Remark 5.13. Acyclicity and bounded tree-width of a conjunctive query are incomparable — all queries $Ex_1x_2 \wedge Ex_2x_3 \wedge \ldots \wedge Ex_{n-1}x_n \wedge Ex_nx_1$, for $n \geq 3$, are of tree-width 2, but cyclic. On the other hand, the queries $R_nx_1 \ldots x_n$, for $n \geq 1$ and n-ary R_n, are acyclic, but their tree-width grows with n.

Chekuri and Rajaraman [7] defined a common generalization of both acyclicity and tree-width they called *query-width*. Acyclic queries are precisely those of query-width 1, and for all queries φ we have query-width$(\varphi) \leq \mathrm{tw}(\mathcal{B}_\varphi)$. Chekuri and Rajaraman [7] showed that conjunctive queries of bounded query-width can be evaluated in polynomial time (in the size of the input and the output).

However, query-width has one big disadvantage: Queries of bounded query-width cannot be recognized in polynomial time. More precisely, for every $q \geq 4$ it is NP-complete to decide whether a given query has query-width at most q [12]. Gottlob, Leone, and Scarcello [12] therefore introduced yet another width, the *hypertree-width* of a conjunctive query. Acyclic queries are precisely the queries of hypertree-width 1, and for every query φ we have hypertree-width$(\varphi) \leq$ query-width(φ). Moreover, for every $h \geq 1$ there is a polynomial time algorithm that recognizes the queries of hypertree-width at most h.

Although we do not want to give the details here, the basic idea of query-width and hypertree-width is easy to explain: Suppose that we are given a query φ, a tree-decomposition $(\mathcal{T}, (X_t)_{t\in T})$ of φ, and a mapping λ that associates with every $t \in T$ a set $\lambda(t)$ of atoms of φ such that $X_t \subseteq \bigcup_{\alpha\in\lambda(t)} \mathrm{var}(\alpha)$. Suppose, moreover, that $|\lambda(t)| \leq h$ for all $t \in T$ (very roughly, the hypertree-width of φ is the smallest h such that a tree-decomposition and a mapping λ with these properties exist).

Then to evaluate φ in a structure $\mathcal{A}$, we can proceed as follows: For every $t \in T$ we let

$$\mathcal{P}_t := \mathop{\bowtie}_{\alpha\in\lambda(t)} \pi_{X_t}(\alpha(\mathcal{A})) \bowtie \mathop{\bowtie}_{\substack{\alpha\in\mathrm{at}(\varphi)\\ \mathrm{node}(\alpha)=t}} \alpha(\mathcal{A}).$$

Then $\varphi(\mathcal{A}) = \pi_{\mathrm{free}(\varphi)}\big(\mathop{\bowtie}_{t\in T}\mathcal{P}_t\big)$. Thus we can use Lemmas 5.3 and 5.4 to compute $\varphi(\mathcal{A})$. Observing that $||\mathcal{P}_t|| \leq ||\mathcal{A}||^h$, this yields a polynomial time algorithm.

Let us now try to extend the results on conjunctive queries to larger classes of formulas. A *conjunctive query with negation* is a formula of the form $\exists \bar{y} \bigwedge_{i=1}^{n} \lambda_i$, where $\lambda_1, \ldots, \lambda_n$ are literals, i.e. atomic or negated atomic formulas. We let $\text{at}^+(\varphi)$ ($\text{at}^-(\varphi)$) denote the set of all atoms occurring positively (negatively, resp.) in φ.

The first observation we make is a disappointment: Evaluating conjunctive queries with negation whose hypergraph is acyclic is just as hard as evaluating arbitrary conjunctive queries with negation. To see this, let $\varphi = \exists \bar{y} \bigwedge_{i=1}^{n} \lambda_i$ be an arbitrary conjunctive query with negation of vocabulary τ. Let R be a new $|\text{var}(\varphi)|$-ary relation symbol, $\bar{x}$ a tuple that contains all variables of φ, and $\varphi^* := \exists \bar{y}\big(\neg R\bar{x} \wedge \bigwedge_{i=1}^{n} \lambda_i\big)$. Then $\mathcal{H}_{\varphi^*}$ is acyclic, because the one node tree is a chordal tree-decomposition of $\mathcal{H}_{\varphi^*}$. For every τ-structure $\mathcal{A}$ let $\mathcal{A}^*$ be the $\tau \cup \{R\}$-expansion of $\mathcal{A}$ with $R^{\mathcal{A}^*} := \emptyset$. Then $||\mathcal{A}^*|| = O(||\mathcal{A}||)$ and $\varphi(\mathcal{A}) = \varphi^*(\mathcal{A}^*)$. Thus evaluating φ is not harder than evaluating φ^*.

However, there is a refined notion of acyclicity for conjunctive queries with negation:

Definition 5.14. A conjunctive query with negation φ is *acyclic* if it has a chordal decomposition $(\mathcal{T}, (X_t)_{t\in T})$ such that for every $t \in T$ there is an atom $\alpha \in \text{at}^+(\varphi)$ such that $X_t = \text{var}(\alpha)$.

If φ has such a chordal decomposition that, in addition, is a strict tree-decomposition, then φ is *strictly acyclic*.

Observe that for an acyclic conjunctive query with negation φ, for every $\alpha \in \text{at}^-(\varphi)$ there exists a $\beta \in \text{at}^+(\varphi)$ such that $\text{var}(\alpha) \subseteq \text{var}(\beta)$. Thus every reduced chordal decomposition $(\mathcal{T}, (X_t)_{t\in T})$ of the hypergraph $\mathcal{H}_\varphi$ satisfies the additional condition that for every $t \in T$ there is an atom $\alpha \in \text{at}^+(\varphi)$ such that $X_t = \text{var}(\alpha)$. On the other hand, if φ is not acyclic, then no chordal decomposition of $\mathcal{H}_\varphi$ satisfies this condition.

Proposition 5.15. *There is a linear-time algorithm that decides whether a given conjunctive query with negation is acyclic.*

Proof: Given a conjunctive query with negation φ, the algorithm first computes a reduced chordal decomposition $(\mathcal{T}, (X_t)_{t\in T})$ of the hypergraph $\mathcal{H}_\varphi$ (by Theorem 5.6 and Lemma 3.3). If this is impossible, then φ is not acyclic. Then it checks whether for every node $t \in T$ there is at least one $\alpha \in \text{at}^+(\varphi)$ such that $\text{node}(\alpha) = t$. If this is not the case, then φ is not acyclic. Otherwise, φ is acyclic, and the algorithm returns the chordal decomposition $(\mathcal{T}, (H_t)_{t\in T})$. □

Corollary 5.16. *The evaluation problem for acyclic conjunctive queries with negation can be solved in time* $O\big(||\varphi|| \cdot ||\mathcal{A}|| \cdot ||\varphi(\mathcal{A})||\big)$.

For strictly acyclic queries, this can be improved to $O\big(||\varphi|| \cdot ||\mathcal{A}||\big)$.

Proof: We can proceed completely analogously to the proof of Theorem 5.7; the only problem being that to compute the relations $\mathcal{P}_t$ we now have to form joins with relations $\neg\alpha(\mathcal{A})$, where $\alpha \in \text{at}^-(\mathcal{A})$. For such negative literlas the size of $\neg\alpha(\mathcal{A})$ is no longer bounded by $||\mathcal{A}||$.

However, by acyclicity the following observation gives the desired time bounds: Let $Y \subseteq X$, $\mathcal{R}$ an X-relation and $\mathcal{S}$ a Y relation. Then $\mathcal{R} \bowtie \mathcal{S}^{\complement}$, where $\mathcal{S}^{\complement}$ denotes the complement of $\mathcal{S}$, can be computed in time $O(||\mathcal{R}|| + ||\mathcal{S}||)$. This can be done similarly to computing the join of $\mathcal{R}$ and $\mathcal{S}$. □

For queries of bounded tree-width, we do not get any new problems by allowing negation. Similarly to Corollary 5.10 we can prove:

Corollary 5.17. *The evaluation problem for conjunctive queries with negation can be solved in time* $O\big(||\varphi||^{w+2} + ||\varphi|| \cdot (|A|^{w+1} + ||\mathcal{A}||) \cdot ||\varphi(\mathcal{A})||\big)$, *where* $w := \text{tw}(\varphi)$, *and in time* $O\big(||\varphi||^{s+2} + ||\varphi|| \cdot (|A|^{s+1} + ||\mathcal{A}||)\big)$, *where* $s := \text{stw}(\varphi)$.

5.2. Non-recursive stratified datalog. A *datalog rule (with negation)* ρ is an expression of the form

$$\gamma \leftarrow \lambda_1 \wedge \ldots \wedge \lambda_n,$$

where γ is an atom of the form $Qx_1 \dots x_l$ with pairwise distinct variables $x_1, \dots, x_l \in \text{var}(\lambda_1 \wedge \dots \wedge \lambda_n)$ and $\lambda_1, \dots, \lambda_n$ are literals. γ is called the *head* and $\lambda_1 \wedge \dots \wedge \lambda_n$ the *body* of ρ.

To define the semantics, suppose that $\gamma = Qx_1 \dots x_l$ and let $\mathcal{A}$ be a structure whose vocabulary contains all relation symbols occurring in the body of ρ. Then we let

$$\rho(\mathcal{A}) := \pi_{\{x_1,\dots,x_l\}}\Big(\bowtie_{1 \le i \le n} \lambda_i(\mathcal{A})\Big).$$

Letting $\bar{y}$ be a tuple that consists of all variables in $\bigcup_{1 \le i \le n} \text{var}(\lambda_i) \setminus \{x_1, \dots, x_l\}$ and

$$\varphi(\bar{x}) := \exists \bar{y} \bigwedge_{1 \le i \le n} \lambda_i$$

we have $\rho(\mathcal{A}) = \varphi(\mathcal{A})$. Thus datalog rules are just another way of writing conjunctive queries with negation.

A *datalog program* is a finite set Π of datalog rules. The *intensional vocabulary* $\text{int}(\Pi)$ of a Π is the set of all relation symbols that occur in the head of some rule of Π. The extensional vocabulary $\text{ext}(\Pi)$ is the set of all relation symbols that occur in the body of some rule of Π and are not contained in $\text{int}(\Pi)$. Program Π is *non-recursive*, if no relation symbol that occurs in the head of a rule also occurs in the body of a rule. *In the following we restrict our attention to non-recursive datalog programs.*

To define the semantics, let $Q \in \text{int}(\Pi)$ and $\mathcal{A}$ an $\text{ext}(\Pi)$-structure. Then we let

$$\Pi_Q(\mathcal{A}) := \bigcup \big\{\rho(\mathcal{A}) \bigm| \rho \in \Pi, Q \text{ occurs in the head of } \rho\big\}.$$

It is easy to see that the query $\mathcal{A} \mapsto \Pi_Q(\mathcal{A})$ can be defined by an existential first-order formula, and that conversely every query definable by an existential first-order formula can also be defined by a datalog program.

A *non-recursive stratified datalog (NRSD) program* is a sequence $\Pi := (\Pi^1, \dots, \Pi^n)$ of non-recursive datalog programs with the property that for $1 \le i < j \le n$ we have $\text{int}(\Pi_j) \cap (\text{int}(\Pi_i) \cup \text{ext}(\Pi_i)) = \emptyset$, i.e. no $Q \in \text{int}(\Pi^j)$ occurs in Π_i. The intensional vocabulary of Π is the set $\text{int}(\Pi) := \bigcup_{i=1}^n \text{int}(\Pi_i)$, and the extensional vocabulary is the set

$$\text{ext}(\Pi) := \bigcup_{i=1}^{n} \Big(\text{ext}(\Pi^i) \setminus \bigcup_{j=1}^{i-1} \text{int}(\Pi^j)\Big).$$

The programs $\Pi_1, \dots, \Pi_n$ are called the *strata* of Π.

To define the semantics, let $\mathcal{A}$ be an $\text{ext}(\Pi)$-structure. Inductively over i, $1 \le i \le n$, we define $\Pi_Q(\mathcal{A})$ for all $Q \in \text{int}(\Pi^i)$ and a structure $\mathcal{A}_i$ as follows: As the induction bases, we let $\mathcal{A}_0 := \mathcal{A}$. For the induction step, suppose that $1 \le i \le n$ and that the $\text{ext}(\Pi) \cup \bigcup_{j=1}^{i-1} \text{int}(\Pi^j)$-structure $\mathcal{A}_{i-1}$ is already defined. For $Q \in \text{int}(\Pi^i)$ we let $\Pi_Q(\mathcal{A}) := \Pi^i_Q(\mathcal{A}_{i-1})$. We let $\mathcal{A}_i$ be the $\text{ext}(\Pi) \cup \bigcup_{j=1}^{i} \text{int}(\Pi^j)$-expansion of $\mathcal{A}_{i-1}$ with $Q^{\mathcal{A}_i} := \Pi_Q(\mathcal{A})$ for $Q \in \text{int}(\Pi_i)$.

We say that a query χ of vocabulary τ is *NRSD-definable* if there is an NRSD-program Π with $\text{ext}(\Pi) = \tau$ and a relation symbol $Q \in \text{int}(\Pi)$ such that for all τ-structures $\mathcal{A}$ we have $\chi(\mathcal{A}) = \Pi_Q(\mathcal{A})$.

It is easy to prove the following (well-known) fact:

Fact 5.18. *A query is first-order definable if, and only if, it is NRSD-definable.*

In the following, we assume that every NRSD-program has a distinguished intensional *goal predicate*, which we always denote by Q. Then we write $\Pi(\mathcal{A})$ instead of $\Pi_Q(\mathcal{A})$. An NRSD-program Π is *Boolean*, if it defines a Boolean query. Π is *equivalent* to a formula φ or to another program Π' if they define the same query.

The *evaluation problem* for a class P of datalog programs is the following problem:

Input:	Structure $\mathcal{A}$, program $\Pi \in$ P.
Problem:	Compute $\Pi(\mathcal{A})$.

Tree-decompositions, tree-width, acyclicity, etc. of a datalog rule ρ are defined with respect to the corresponding conjunctive query with negation. For example, a tree-decomposition of a datalog rule $\rho := \gamma \leftarrow \bigwedge_{i=1}^{n} \lambda_i$ is a tree-decomposition of the hypergraph $(\text{var}(\rho), \{\text{var}(\lambda_i) \mid 1 \leq i \leq n\})$, and a tree-decomposition $(\mathcal{T}, (X_t)_{t\in T})$ of ρ is *strict* if there is a $t \in T$ such that $\text{var}(\gamma) \subseteq X_t$.

Definition 5.19. Let $\Pi = (\Pi_1, \dots, \Pi_n)$ be an NRSD-program.

(1) Π is *strictly acyclic* if every rule of Π is strictly acyclic.

(2) Π is *acyclic* if $(\Pi_1, \dots, \Pi_{n-1})$ is strictly acyclic and every rule ρ of Π_n is acyclic.

(3) The *strict tree-width* of Π is the number

$$\text{stw}(\Pi) := \max\{\text{stw}(\rho) \mid \rho \in \bigcup_{i=1}^{n} \Pi_i\}.$$

(4) The *tree-width* of Π is the number

$$\text{tw}(\Pi) := \max\{\text{stw}((\Pi_1, \dots, \Pi_{n-1}))\} \cup \{\text{tw}(\rho) \mid \rho \in \Pi_n\}.$$

If we make the (natural) assumption that the goal predicate is the only intensional symbol of an NRSD-program, then the notions of acyclicity and strict acyclicity, and similarly tree-width and strict tree-width, coincide for Boolean and unary programs (i.e. for programs whose goal predicate is at most unary).

The following example shows why it is necessary that in an acyclic program we require all strata except for the last one to be *strictly* acyclic:

Example 5.20. Let $\Pi = (\Pi_1, \dots, \Pi_n)$ be an arbitrary NRSD-program. We construct an equivalent program $\Pi' = (\Pi'_0, \Pi'_1, \dots, \Pi'_n)$ such that all rules of Π' are acyclic.

Let m be the maximal number of variables occurring in a rule of Π and $X \notin \text{int}(\Pi) \cup \text{ext}(\Pi)$ a new m-ary relation symbol. Let ρ_0 be the acyclic datalog rule

$$Xx_1 \dots x_m \leftarrow \bigwedge_{i=1}^{m} x_i = x_i.$$

Π'_0 just consists of the rule ρ_0, and for $1 \leq i \leq n$ the stratum Π'_i is obtained from Π_i by adding an atom $X\bar{y}$ to the body of every rule ρ of Π_i, where $\bar{y}$ is a tuple of variables that contains all variables of ρ.

Clearly, Π' is equivalent to Π, and all of its rules are acyclic.

Definition 5.21. An NRSD-program $\Pi = (\Pi_1, \dots, \Pi_n)$ is in *normal form* if for $1 \leq i < j \leq n$ and $X \in \text{int}(\Pi_i) \cap \text{ext}(\Pi_j)$ the relation symbol X only occurs negatively in Π_j.

Lemma 5.22. *There is an exponential time algorithm that associates with every acyclic NRSD-program* Π *an equivalent acyclic NRSD-program* Π' *in normal form.*

Furthermore, if Π *is strictly acyclic, then* Π' *is also strictly acyclic.*

Proof: Let Π be an acyclic NRSD-program. The *weight* of a symbol $X \in \text{int}(\Pi)$ is the maximal number of positive occurrences of X in the body of a rule.

The following elimination step reduces the weight of an $X \in \text{int}(\Pi)$. Suppose that the rules of Π with X in their head are $X\bar{x} \leftarrow \beta_1(\bar{x}, \bar{y}_1), \dots, X\bar{x} \leftarrow \beta_m(\bar{x}, \bar{y}_m)$. Then for every rule ρ of Π such that X occurs positively in the body of ρ we do the following: Let $\rho := \gamma \leftarrow X\bar{w} \wedge \beta(\bar{w}, \bar{z})$. Without loss of generality we can assume that the tuples $\bar{z}$ and $\bar{w}$ are disjoint from $\bar{y}_1, \dots, \bar{y}_m$. We replace ρ by the rules $\gamma \leftarrow \beta_1(\bar{w}, \bar{y}_1) \wedge \beta(\bar{w}, \bar{z}), \dots, \gamma \leftarrow \beta_m(\bar{w}, \bar{y}_m) \wedge \beta(\bar{w}, \bar{z})$. The resulting program is equivalent to Π. It is easy to see that the new program is (strictly) acyclic if Π is.

After we have done this for all rules ρ such that X occurs positively in the body of ρ, we have reduced the weight of X by 1. Furthermore, if $\beta_1, \dots, \beta_m$ (i.e. the bodies of the rules with X in their head) do not contain any positive occurrence of another intensional symbol of Π, then we have not increased the weight of any other symbol.

We repeat the elimination step (for all symbols of positive weight) until we obtain a program in which all intensional relation symbols have weight 0. To guarantee that we never increase the weight of any symbol, we process the symbols in the order of the strata in which they occur as intensional symbols.

Then the number of elimination steps needed is bound by $||\Pi||$. Each elimination step can be performed in polynomial time, and it at most squares the size of the program. Thus our algorithm can be implemented as an exponential time algorithm. $\square$

The results of Section 5.1 on conjunctive queries yield:

Corollary 5.23. *(1) The evaluation problem for acyclic NRSD-programs in normal form can be solved in time $O\big(||\Pi|| \cdot ||\mathcal{A}|| \cdot ||\Pi(\mathcal{A})||\big)$.*

For strictly acyclic NRSD-programs in normal form, this can be improved to $O\big(||\Pi|| \cdot ||\mathcal{A}||\big)$.

(2) The evaluation problem for NRSD-programs can be solved in time $O\big(||\Pi||^{w+2} + ||\Pi|| \cdot (|A|^{w+1} + ||\mathcal{A}||) \cdot (||\Pi(\mathcal{A})|| + 1)\big)$, where $w := \mathrm{tw}(\Pi)$, and in time $O\big(||\Pi||^{s+2} + ||\Pi|| \cdot (|A|^{s+1} + ||\mathcal{A}||)\big)$, where $s := \mathrm{stw}(\Pi)$.

The following example shows why we need the normal form in (1). It also shows that there is no polynomial time translation of arbitrary acyclic programs into normal form programs. A similar example occurs in [10].

Example 5.24. Let $n \geq 2$ and $X_1, \dots, X_n$ be n-ary relation symbols. For $1 \leq i < j \leq n, 1 \leq k \leq n-1$ let $\rho^k_{\{i,j\}}$ be the datalog rule

$$X_{k+1}x_1 \dots x_{i-1}x_jx_{i+1} \dots x_{j-1}x_ix_{j+1} \dots x_n \leftarrow X_kx_1 \dots x_n.$$

Furthermore, let ρ^k_{id} be the rule $X_{k+1}\bar{x} \leftarrow X_k\bar{x}$ and $\Pi_k := \{\rho^k_{\mathrm{id}}\} \cup \{\rho^k_{\{i,j\}} \mid 1 \leq i < j \leq n-1\}$. Then $\Pi = (\Pi_1, \dots, \Pi_{n-1})$ is a strictly acyclic NRSD-program with $||\Pi|| = O(n^4)$.

Now let $\mathcal{A}$ be the $\{X_1\}$-structure $(\{1, \dots, n\}, \{(1, \dots, n)\})$. Then, using the fact that every permutation of $\{1, \dots, n\}$ is the product of at most $(n-1)$ transpositions, we obtain

$$\Pi_{X_n}(\mathcal{A}) = \big\{(x_1, \dots, x_n) \bigm| \{x_1, \dots, x_n\} = \{1, \dots, n\}\big\}.$$

Thus $|\Pi_{X_n}(\mathcal{A})| = n!$, and therefore $\Pi_{X_n}(\mathcal{A})$ cannot be computed in time $O(||\Pi|| \cdot ||\mathcal{A}||) = O(n^5)$.

Remark 5.25. Similar results can be obtained for stratified datalog programs with recursion (by iterating the results for programs without recursion).

5.3. Tractable fragments of first-order logic. In this last subsection we show that the classes of strictly acyclic NRSD-programs and NRSD-programs of strictly bounded tree-width correspond to well-known fragments of first-order logic.

The guarded fragment. The *guarded fragment* GF is the smallest fragment of FO such that:

(1) GF contains all atoms.

(2) If $\varphi, \varphi_1, \varphi_2$ are GF-formulas, then so are $\neg\varphi$, $\varphi_1 \wedge \varphi_2$, and $\varphi_1 \vee \varphi_2$.

(3) If α is atomic and φ is a GF-formula with $\mathrm{free}(\varphi) \subseteq \mathrm{var}(\alpha)$, then for every tuple $\bar{y}$ of variables $\exists\bar{y}(\alpha \wedge \varphi)$ is a GF-formula.

As usual we use the Boolean connectives $\rightarrow, \leftrightarrow$ and the universal quantifier as abbreviations.

A GF-formula is *strictly guarded* if it is of the form $\exists\bar{y}(\alpha \wedge \varphi)$, where $\mathrm{free}(\varphi) \subseteq \mathrm{var}(\alpha)$. Here we allow the degenerated case that $\bar{y}$ is empty, i.e. that the formula is just $\alpha \wedge \varphi$. Every GF-formula is a Boolean combination of atomic formulas and strictly guarded formulas. Furthermore, any GF-sentence φ is equivalent to the strictly guarded sentence $\exists y(y = y \wedge \varphi)$.

Theorem 5.26. *(1) There is a quadratic time algorithm that associates with every strictly guarded formula an equivalent strictly acyclic NRSD-program in normal form.*

(2) There is an algorithm that associates with every strictly acyclic NRSD-program an equivalent disjunction of strictly guarded formulas.

In particular, every GF-sentence is equivalent to a Boolean acyclic NRSD-program and vice versa.

Proof: To prove (1), we note first that every strictly guarded formula can easily be translated to a strictly guarded GF-formula in which negation symbols only appear directly in front of atoms or existential quantifiers. Let φ be such a formula.

To translate φ, we proceed by induction. Assume first that $\psi := \exists \bar{y}(\alpha \wedge \beta)$ is a strictly guarded subformula of φ with a quantifier-free β. Using the equality $\alpha \wedge (\beta_1 \vee \beta_2) \equiv (\alpha \wedge \beta_1) \vee (\alpha \wedge \beta_2)$, we can easily transform ψ to an equivalent datalog program Π every rule of which contains α. Since $\mathrm{var}(\alpha) \supseteq \mathrm{var}(\beta)$, this program is strictly acyclic. Furthermore, since we need at most one copy of α for every literal in β, the length of the program is at most quadratic in the length of φ.

We can now use induction to translate arbitrary strictly guarded subformulas of φ.

To prove (2), we show that every strictly acyclic datalog rule ρ can be translated to an equivalent strictly guarded formula in linear time. Then a straightforward induction using Lemma 5.22 yields the full statement.

Consider the rule $\rho := \gamma \leftarrow \lambda_1 \wedge \ldots \wedge \lambda_n$ and let $(\mathcal{T}, (X_t)_{t \in T})$ be a reduced strict chordal decomposition of ρ. Without loss of generality we can assume that $\mathrm{var}(\gamma) \subseteq X_{r^{\mathcal{T}}}$.

For the root $r := r^{\mathcal{T}}$, let $Z_r := \mathrm{var}(\gamma)$. For every node $t \in T \setminus \{r\}$ with parent s, let $Z_t := X_t \cap X_s$. For every $t \in T$, let $\bar{z}_t$ a tuple of variables that contains all variables in Z_t and $\bar{y}_t$ a tuple that contains all variables in $X_t \setminus Z_t$. If $t_1, \ldots, t_m$ are the children of t, let

$$\varphi_t(\bar{z}_t) := \exists \bar{y}_t \Big(\bigwedge_{\substack{1 \leq i \leq n \\ \mathrm{node}(\lambda_i) = t}} \lambda_i \wedge \bigwedge_{j=1}^{m} \varphi_{t_j}(\bar{z}_{t_j}) \Big).$$

Then a straightforward induction shows that ρ is equivalent to φ_r. Moreover, every φ_t is strictly guarded since the conjunction $\bigwedge_{\substack{1 \leq i \leq n \\ \mathrm{node}(\lambda_i) = t}} \lambda_i$ contains an atom λ_{i_t} with $\mathrm{var}(\lambda_{i_t}) = X_t$, and the free variables of all other conjuncts of φ_t are contained in X_t. □

Remark 5.27. It seems possible to improve the translation algorithm in (1) to an $O(n \cdot \log(n))$-algorithm. We do not believe that it can be made linear, although we cannot prove this. The following family $(\varphi_n)_{n \geq 1}$ of strictly acyclic formulas seem to require NRSD-programs of superlinear size: We let $P_1, P_2, \ldots$ be unary and, for $n \geq 1$, R_n n-ary. Then we let

$$\varphi_n := R_n x_1 \ldots x_n \wedge \bigvee_{i=1}^{n} P_i x_i.$$

The translation from NRSD-programs to first-order formulas cannot be made polynomial; this has nothing to do with being acyclic or guarded. Intuitively, it is due to the fact that programs correspond to certain directed acyclic graphs, whereas formulas correspond to trees.

Remark 5.28. Gottlob, Grädel, and Veith [10] give a similar translation between sentences of the guarded fragment and so-called *Datalog LITE*-programs, which are essentially stratified datalog programs where every rule has a strict chordal decomposition whose tree only consists of one node.

Finite-variable fragments. For $k \geq 1$ we let FO^k denote the set of all first-order formulas with at most k variables.

Theorem 5.29. *Let $k \geq 1$.*

(1) There is a linear time algorithm that associates with every FO^k*-formula an equivalent NRSD-program of strict tree-width at most* $(k-1)$.

(2) There is an algorithm that associates with every NRSD-program of strict tree-width at most $(k-1)$ *an equivalent* FO^k*-formula.*

Proof: (1) If we translate an FO^k-formula to an NRSD-program in a straightforward manner, then every rule of the resulting program has at most k-variables and thus strict tree-width at most $(k-1)$.

(2) It suffices to prove that every conjunctive query of strict tree-width at most $k-1$ is equivalent to an FO^k-formula. This has been done by Kolaitis and Vardi [13]; the proof is similar to the proof that every strictly acyclic datalog rule is equivalent to a strictly guarded formula. It uses the fact that a first-order formula φ with the property that all subformulas of φ have at most k free variables can be transferred to an equivalent FO^k-formula by renaming variables. □

6 The overall picture

The query-evaluation algorithms of Sections 4 and 5 are very similar. First we compute a tree-decomposition, either of the structure or the formula. Then we pass the tree three times. The first (bottom-up) pass is to compute all reachable states of the automaton at some tree-node (the sets P_t in the proof of Theorem 4.2) or all reachable assignments to the variables occurring at the node (the relation $\mathcal{Q}_t$ in the proof of Lemma 5.3). In the second (top-down) pass we filter out all states that do not lead to an accepting configuration (S_t in the proof of Theorem 4.2, $\mathcal{R}_t$ in Lemma 5.3). The third (bottom-up) pass is to assemble the satisfying assignments from the pieces computed at every node ($\mathrm{Sat}_{t,q}$ in the proof of Theorem 4.2, $\mathcal{S}_t$ in Lemma 5.4). Note that in both cases for sentences we only need the first pass.

The connection becomes clearer if in Section 4 we view the structures as automata: For every τ-structure $\mathcal{A}$ we define a non-deterministic[2] tree-automaton $\mathfrak{A}_{\mathcal{A}} := (Q, \delta, \Delta, F)$. Let r be the arity of τ. The alphabet is $\Sigma_\tau := \tau \times \mathrm{Pow}(\{(i,j) \mid 1 \leq i,j \leq r\}) \times \mathrm{Pow}(\{(i,j) \mid 1 \leq i,j \leq r\})$. The state space consists of a state $q^{=}_{(a,a)}$ for every $a \in A$ and a state $q^R_{\bar{a}}$ for every $R \in \tau, \bar{a} \in R^{\mathcal{A}}$. In the following we just view '$=$' as a relation symbol in τ with $=^{\mathcal{A}}:= \{(a,a) \mid a \in A\}$. The transition relation δ consists of all tuples $(q^R_{\bar{a}}, q^{R'}_{\bar{a}'}, (R'', e, e'), q^{R''}_{\bar{a}''})$ where $\big((i,j) \in e \implies a_i = a''_j\big)$ and $\big((i,j) \in e' \implies a'_i = a''_j\big)$. The starting relation Δ consists of all pairs $((R, \emptyset, \emptyset), q^R_{\bar{a}})$. Every state of $\mathfrak{A}_{\mathcal{A}}$ is accepting, i.e. we let $F := Q$.

Now let φ be an acyclic conjunctive query of vocabulary τ. Then, starting from an arbitrary chordal decomposition of φ, we can find a chordal decomposition $(\mathcal{T}, (X_t))$ of φ where $\mathcal{T}$ is a binary tree, together with an onto mapping $\lambda : T \to \mathrm{at}(\varphi)$ such that for every $t \in T$ we have $\mathrm{var}(\lambda(t)) = X_t$. We define a mapping $\sigma : T \to \Sigma_\tau$ as follows: For the leaves $t \in T$ we let $\sigma(t) = (R, \emptyset, \emptyset)$, where R is the relation symbol occurring in $\lambda(t)$. For a node t with children u_1 and u_2 we let $\sigma(t) = (R, e_1, e_2)$, where again R is the relation symbol occurring in $\lambda(t)$, and e_i is defined as follows (for $i = 1, 2$): If $\lambda(t) = Rx_1 \ldots x_m$ and $\lambda(u_i) = R'y_1 \ldots y_n$ (note that $1 \leq m, n \leq r$), then $e = \{(k,l) \mid y_k = x_l\}$.

Suppose now that φ is a sentence. Then the automaton $\mathfrak{A}_{\mathcal{A}}$ accepts the colored tree $(\mathcal{T}, \gamma)$ if, and only if, $\mathcal{A} \models \varphi$. The fact that our automaton is not deterministic does not play a role as long as we just want to decide acceptance; in the algorithm of Theorem 4.2 we used the fact that there we had a deterministic automaton only in the third pass in order to compute the output efficiently.

The correspondence breaks down for formulas with free variables. If we want to evaluate an MSO-formula on a structure of bounded tree-width then in the third pass of the automaton we have to collect additional colorings of the tree that lead to accepting runs. If we want to evaluate an acyclic conjunctive query in a structure $\mathcal{A}$ using the automaton $\mathfrak{A}_{\mathcal{A}}$, then in the third pass we have to collect projections of accepting runs of the automaton.

If we restrict our attention to monadic second-order sentences of a very special form, there is an abstract explanation for the connection between evaluation of the MSO-sentences and Boolean conjunctive queries due to Feder and Vardi [9]: Both problems amount to deciding whether there is a homomorphism between two relational structures. And in both cases we use the fact that it is decidable in polynomial time if an arbitrary structure $\mathcal{A}$ contains a homomorphic image of a structure $\mathcal{B}$ of bounded tree-width.

[2]In Section 4.1 we only defined deterministic tree-automata. In a non-deterministic automaton (Q, δ, Δ, F) of alphabet Σ, instead of functions we have a *transition relation* $\delta \subseteq Q \times Q \times \Sigma \times Q$ and a *starting relation* $\Delta \subseteq \Sigma \times Q$.

Appendix: Basic algorithmic routines

In this appendix we describe how to implement the basic algorithmic routines needed in Section 5 — computing joins and projections — as linear time algorithms on a standard RAM. We are sure these algorithms are known, but since we could not find a reference, we decided to add this appendix.

Recall the definitions of an X-relation and the projection and join operations from page 13. Let $\mathcal{R}$ be an X-relation. As usual, we assume that both X and range$(\mathcal{R})$ are sets of natural numbers. $\mathcal{R}$ is represented by a *header* together with all tuples in the relation. The header consists of $|X|+2$ numbers: The cardinality of X, the cardinality of $\mathcal{R}$, and then an enumeration of X. Every tuple in the relation is represented by $|X|$ numbers. We let $||\mathcal{R}|| := 2 + |X| + |X| \cdot |\mathcal{R}|$. For every $x \in X$ we let $n_x := n_x(\mathcal{R}) := \max\big(\pi_{\{x\}}(\mathcal{R})\big)$.

For every linear order $\preceq$ on a subset Y of X, we can define the *lexicographic (pre-)order with respect to* $\preceq$ on the set of all mappings $X \to \mathbb{N}$ by letting $\gamma \preceq \delta$ if, and only if, either $\gamma|_Y = \delta|_Y$ or there is a $y \in Y$ such that $\gamma(y) \prec \delta(y)$ and $\gamma(z) = \delta(z)$ for all $z \in Y$ such that $z \prec y$.

Using the well-known bucket sort algorithm, we obtain the following:

Lemma A.1. *Let $\mathcal{R}$ be an X-relation and $\preceq$ a linear order on a subset Y of X. Then $\mathcal{R}$ can be ordered lexicographically with respect to $\preceq$ in time $O(||\mathcal{R}|| + \sum_{y\in Y} n_y)$.*

Once we have sorted the relations, it is easy to compute projections and joins:

Lemma A.2. *(1) Let $\mathcal{R}$ be an X-relation and $Y \subseteq X$. Then $\pi_Y(\mathcal{R})$ can be computed in time $O(||\mathcal{R}|| + \sum_{y\in Y} n_y)$.*

(2) Let $\mathcal{R}$ be an X-relation and $\mathcal{S}$ a Y-relation. Then $\mathcal{R} \bowtie \mathcal{S}$ can be computed in time $O\Big(||\mathcal{R}|| + ||\mathcal{S}|| + ||\mathcal{R} \bowtie \mathcal{S}|| + \sum_{z\in X\cap Y} \big(n_z(\mathcal{R}) + n_z(\mathcal{S})\big)\Big)$.

Proof: (1) We first compute the projection in a straightforward way by copying the Y-part of every tuple in $\mathcal{R}$ to a new list. Then we sort this new list lexicographically with respect to the natural order on Y and eliminate double entries.

(2) We sort both relations lexicographically with respect to the natural order on $X \cap Y$. It is straightforward to compute the join of the ordered relations in time $O(||\mathcal{R}|| + ||\mathcal{S}|| + ||\mathcal{R} \bowtie \mathcal{S}||)$. □

Lemma A.2 gives us good time bounds for computing projections and joins as long as we are dealing with X-relations $\mathcal{R}$ where $\sum_{x\in X} n_x = O(\mathcal{R})$. For arbitrary X-relations, we can improve these algorithms by first "compressing" the relations.

A *compression* of a set N of natural numbers is an injective mapping $f : N \to \{1, \dots, |N|\}$. A compression is represented by two arrays A_f and $A_{f^{-1}}$. The size of A_f is max(N), and its ith entry is $f(i)$ if $i \in N$ and 0 otherwise. The size of $A_{f^{-1}}$ is $|N|$, and its ith entry is $f^{-1}(i)$.

Lemma A.3. *Given a set N of natural numbers, a compression of N (represented as described above) can be computed in linear time.*

Proof: Recalling that in the RAM model all memory cells are initialized to '0', the proof is straightforward. □

Remark A.4. Although the last lemma is formally correct, we somehow seem to be cheating, for two reasons:

The first is that we assume all memory cells to be initialized to '0'. It is easy to avoid this assumption. Suppose initially the state of the memory cells is arbitrary. We first initialize the array $A_{f^{-1}}$ by setting all its memory cells to 0, this only requires time $O(|N|)$. Then we can compute the arrays A_f and $A_{f^{-1}}$ as we did before. All entries except those entries of $A_f(i)$ for $i \notin N$ are correct. However, we can still find out if $A_f(i)$ is supposed to be zero. Say, we read $A_f(i) = j$. Then we check if $A_{f^{-1}}(j) = i$. If this is the case, then j is correct, if not it is supposed to be zero. This method is essentially the "lazy array" technique described in [15].

The second problem is that $\max(N)$ may be much larger than $|N|$, so we waste a lot of space with the table A_f. In practice, we may have to use a hash table to represent A_f more space efficient. This gives us an additional time factor, but can still be implemented quite efficiently.

Theorem A.5. *(1) Let $\mathcal{R}$ be an X-relation and $Y \subseteq X$. Then $\pi_Y(\mathcal{R})$ can be computed in time $O(||\mathcal{R}||)$.*

(2) Let $\mathcal{R}$ be an X-relation and $\mathcal{S}$ a Y-relation. Then $\mathcal{R} \bowtie \mathcal{S}$ can be computed in time $O(||\mathcal{R}|| + ||\mathcal{S}|| + ||\mathcal{R} \bowtie \mathcal{S}||)$.

Proof: We only prove (2); (1) can be treated analogously.

We are given an X-relation $\mathcal{R}$ and a Y-relation $\mathcal{S}$. Let $Z := X \cap Y$. Our algorithm first computes, for all $z \in Z$, a compression f_z of $\text{range}(\pi_{\{z\}}(\mathcal{R})) \cup \text{range}(\pi_{\{z\}}(\mathcal{S}))$. Then, using the tables A_{f_z}, it computes the relations

$$\mathcal{R}' := \Big\{ \gamma' \;\Big|\; \exists \gamma \in \mathcal{R} : \; \big(\gamma'(z) = f_z(\gamma(z)) \text{ for } z \in Z, \gamma'(x) = \gamma(x) \text{ for } x \in X \setminus Z\big) \Big\},$$
$$\mathcal{S}' := \Big\{ \delta' \;\Big|\; \exists \delta \in \mathcal{R} : \; \big(\delta'(z) = f_z(\delta(z)) \text{ for } z \in Z, \delta'(y) = \delta(y) \text{ for } y \in Y \setminus Z\big) \Big\}$$

This requires time $O(||\mathcal{R}|| + ||\mathcal{S}||)$. Note that $||\mathcal{R}|| = ||\mathcal{R}'||$, $||\mathcal{S}|| = ||\mathcal{S}'||$, and $||\mathcal{R}' \bowtie \mathcal{S}'|| = ||\mathcal{R} \bowtie \mathcal{S}||$. Furthermore, for all $z \in Z$ we have $n_z(\mathcal{R}'), n_z(\mathcal{S}') \leq |\pi_{\{z\}}(\mathcal{R}') \cup \pi_{\{z\}}(\mathcal{S}')| \leq |\mathcal{R}'| + |\mathcal{S}'|$ and thus $\sum_{z \in Z} \big(n_z(\mathcal{R}') + n_z(\mathcal{S}')\big) = O\big(||\mathcal{R}'|| + ||\mathcal{S}'||\big)$.

Thus by Lemma A.2, we can compute $\mathcal{R}' \bowtie \mathcal{S}'$ in time $O(||\mathcal{R}|| + ||\mathcal{S}|| + ||\mathcal{R} \bowtie \mathcal{S}||)$. We can now use the tables $A_{f_z^{-1}}$ to compute $\mathcal{R} \bowtie \mathcal{S}$ from $\mathcal{R}' \bowtie \mathcal{S}'$ in time $O(||\mathcal{R} \bowtie \mathcal{S}||)$. $\square$

Acknowledgements. We thank Helmut Veith for pointing us to the "lazy arrays" in [15].

References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[3] H. Andréka, J. van Benthem, and I. Németi. Modal languages and bounded fragments of first-order logic, 1996. ILLC Research Report ML-96-03, University of Amsterdam.

[4] S. Arnborg, D. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic Discrete Methods*, 8:277–284, 1987.

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

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

[7] Ch. Chekuri and A. Rajaraman. Conjunctive query containment revisited. In Ph. Kolaitis and F. Afrati, editors, *Proceedings of the 5th International Conference on Database Theory*, volume 1186 of *Lecture Notes in Computer Science*, pages 56–70. Springer-Verlag, 1997.

[8] B. Courcelle. Graph rewriting: An algebraic and logic approach. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume 2, pages 194–242. Elsevier Science Publishers, 1990.

[9] T. Feder and M.Y. Vardi. Monotone monadic SNP and constraint satisfaction. In *Proceedings of the 25th ACM Symposium on Theory of Computing*, pages 612–622, 1993.

[10] G. Gottlob, E. Grädel, and H. Veith. Datalog lite: Temporal versus deductive reasoning in verification. Technical Report DBAI-TR-98-22, Technische Universität Wien, 1998.

[11] G. Gottlob, N. Leone, and F. Scarcello. The complexity of acyclic conjunctive queries. In *Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science*, pages 706–715, 1998.

[12] G. Gottlob, N. Leone, and F. Scarcello. Hypertree decompositions and tractable queries. In *Proceedings of the 18th ACM Symposium on Principles of Database Systems*, pages 21–32, 1999.

[13] Ph.G. Kolaitis and M.Y. Vardi. Conjunctive-query containment and constraint satisfaction. In *Proceedings of the 17th ACM Symposium on Principles of Database Systems*, pages 205–213, 1998.

[14] J. Makowsky. Model theory in computer science: An appetizer. In S. Abramsky, D.M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 1, chapter 6. Oxford University Press, 1992.

[15] B. Moret and H. Shapiro. *Algorithms from P to NP*. Benjamin/Cummings, 1990.

[16] L.J. Stockmeyer. *The Complexity of Decision Problems in Automata Theory*. PhD thesis, Department of Electrical Engineering, MIT, 1974.

[17] R.E. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal on Computing*, 13:566–579, 1984.

[18] C. Thomassen. Embeddings and minors. In R. Graham, M. Grötschel, and L. Lovász, editors, *Handbook of Combinatorics*, volume 1, chapter 5, pages 301–349. Elsevier, 1995.

[19] P. van Emde Boas. Machine models and simulations. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume 1, pages 1–66. Elsevier Science Publishers, 1990.

[20] M. Y. Vardi. The complexity of relational query languages. In *Proceedings of the 14th ACM Symposium on Theory of Computing*, pages 137–146, 1982.

[21] M. Y. Vardi. On the complexity of bounded-variable queries. In *Proceedings of the 14th ACM Symposium on Principles of Database Systems*, pages 266–276, 1995.

[22] M. Yannakakis. Algorithms for acyclic database schemes. In *7th International Conference on Very Large Data Bases*, pages 82–94, 1981.