Counting and Enumeration Problems with Bounded Treewidth[⋆]

Reinhard Pichler, Stefan Rümmele, and Stefan Woltran

Vienna University of Technology, Vienna, Austria
{pichler, ruemmele, woltran}@dbai.tuwien.ac.at

Abstract. By Courcelle's Theorem we know that any property of finite structures definable in monadic second-order logic (MSO) becomes tractable over structures with bounded treewidth. This result was extended to counting problems by Arnborg et al. and to enumeration problems by Flum et al. Despite the undisputed importance of these results for proving fixed-parameter tractability, they do not directly yield implementable algorithms. Recently, Gottlob et al. presented a new approach using monadic datalog to close the gap between theoretical tractability and practical computability for MSO-definable decision problems. In the current work we show how counting and enumeration problems can be tackled by an appropriate extension of the datalog approach.

1 Introduction

The most common problem type studied in algorithms and complexity theory is the class of *decision problems*, which usually ask whether a solution to a given problem instance exists, e.g., whether a given graph has a valid 3-coloring. On the other hand, *counting problems* ask how many solutions an instance possesses, e.g., how many different 3-colorings are possible in the given graph. Finally, *enumeration problems* require as an answer the output of all possible solutions, e.g., all possible 3-colorings of the graph. Unfortunately, many interesting decision problems are computationally intractable. Clearly, the corresponding counting and enumeration problems then are intractable as well. A promising approach for dealing with intractability comes from the area of *parameterized complexity theory* (see [9, 12] for an overview). Thereby the complexity analysis is not only based on the input size but also on some additional (structural) property of the input, the *parameter*. Imagine that some problem admits an algorithm with running time $f(k) \cdot n^{\mathcal{O}(1)}$, where n is the input size, k is the parameter and f is some arbitrary (usually exponential) function. Problems which can be solved by such algorithms are called *fixed-parameter tractable* (FPT). The basic idea is that the running time of those algorithms is still feasible, as long as k remains sufficiently small. The *treewidth* of an input structure, which measures the degree of cyclicity (see Section 2 for a formal definition), is a commonly studied parameter.

Courcelle's Theorem [7] states that every decision problem definable in MSO is FPT (in fact, even linear in the input size) when parameterized by the treewidth of the input structure. This result was extended to counting problems by Arnborg et al. [2] as well

⋆ Supported by the Austrian Science Fund (FWF), project P20704-N18.

as to enumeration problems by Flum et al. [11]. Moreover, Bagan [3] and Courcelle [8] showed that for enumeration problems, this result can be refined by stating that the delay between the output of two successive solutions is fixed-parameter linear. Those proofs are constructive in the sense that they transform the problem of evaluating an MSO formula into a tree language recognition problem, which is then solved via a finite tree automaton (FTA). Although those results are very useful for verifying that a given problem is FPT, they do not help in finding algorithms that are usable in practice, since even very simple MSO formulae quickly lead to a "state explosion" of the FTA [13]. Consequently, it was already stated in [15] that the algorithms derived via Courcelle's Theorem are "useless for practical applications". This creates the need for other tools that help the algorithm designer developing FPT algorithms for specific problems.

An alternative approach using monadic datalog was presented by Gottlob et al. [14]. They proved that for decision problems, the MSO evaluation problem can be transformed into a monadic datalog program, of which the evaluation is FPT (and even linear in the input size). Although the general transformation presented there does not lead to a better running time than the MSO to FTA transformation, it has been shown, that this datalog framework helps to find algorithms for specific problems that are indeed feasible in practice [14]. In [17], datalog was already used in an ad hoc manner to solve some MSO-definable counting problems (including #SAT – the problem of counting the number of satisfying truth assignments of a given propositional formula). However, it remained an open question, whether there exists an appropriate extension of monadic datalog, that is capable of solving *every* MSO-definable counting problem. Moreover, enumeration problems have been completely left out so far.

The goal of the current work is to systematically extend the datalog approach from [14] to counting and enumeration problems in order to give an affirmative answer to the question mentioned above. We identify a nontrivial extension of monadic datalog, which we call *quasi-guarded fragment of extended datalog* and show that this fragment allows us to solve every MSO-definable counting problem over structures with bounded treewidth. As a by-product, these extended datalog programs generate intermediate results which can be exploited by a post-processing algorithm to solve the corresponding enumeration problem.

As for the complexity, we prove a fixed-parameter linear time bound for our counting algorithms (assuming unit cost for arithmetic operations) and a delay between two successive solutions for the enumeration algorithms, that is fixed-parameter linear in the size of the input. Note that Bagan [3] and Courcelle [8] presented enumeration algorithms having a delay that is fixed-parameter linear in the size of the next solution. But since these approaches depend on an MSO to FTA transformation, their usefulness for solving concrete problems in practice is restricted for the same reason as stated for Courcelle's Theorem above.

Results. Our main contributions are:

- *Counting.* We identify an appropriate extension of datalog, capable of expressing every MSO-definable counting problem. Algorithms based on our MSO to datalog transformation solve these counting problems in fixed-parameter linear time, parameterized by the treewidth of the input structure (assuming unit cost for arithmetic operations).

– *Enumeration.* Building upon our algorithms for counting problems, we devise a post-processing method which solves the corresponding MSO-definable enumeration problems. We thus get an algorithm which outputs the results with fixed-parameter linear delay, parameterized by the treewidth of the input structure (assuming unit cost for arithmetic operations).

The paper is organized as follows. After recalling basic notations and results in Section 2, we prove the main result regarding MSO-definable counting problems in Section 3. In Section 4, we present a novel post-processing algorithm, which solves the corresponding enumeration problem. In Section 5, we illustrate our approach by the example of 3-COLORABILITY. A conclusion is given in Section 6.

2 Preliminaries

Finite Structures and Treewidth. A *(relational) signature* $\sigma = \{R_1, \ldots, R_n\}$ is a set of relation (or predicate) symbols. Each relation symbol $R \in \sigma$ has an associated arity $\text{arity}(R) \geq 1$. A *finite structure* $\mathcal{A}$ over signature σ (or simply a σ-structure) consists of a finite *domain* $A = \text{dom}(\mathcal{A})$ plus a relation $R^{\mathcal{A}} \subseteq A^{\text{arity}(R)}$ for each $R \in \sigma$.

A *tree decomposition* $\mathcal{T}$ of a σ-structure $\mathcal{A}$ is a pair (T, χ), where T is a tree and χ maps each node n of T (we use $n \in T$ as a shorthand below) to a *bag* $\chi(n) \subseteq \text{dom}(\mathcal{A}) = A$ with the following properties: (1) For each $a \in A$, there is an $n \in T$, s.t. $a \in \chi(n)$. (2) For each $R \in \sigma$ and each $(a_1, \ldots, a_\alpha) \in R^{\mathcal{A}}$, there is an $n \in T$, s.t. $\{a_1, \ldots, a_\alpha\} \subseteq \chi(n)$. (3) For each $n_1, n_2, n_3 \in T$, s.t. n_2 lies on the path from n_1 to n_3, $\chi(n_1) \cap \chi(n_3) \subseteq \chi(n_2)$ holds. The third condition is usually referred to as the *connectedness condition*. The *width* of a tree decomposition $\mathcal{T} = (T, \chi)$ is defined as $\max\{|\chi(n)| \mid n \in T\} - 1$. The *treewidth* of $\mathcal{A}$, denoted as $\text{tw}(\mathcal{A})$, is the minimal width of all tree decompositions of $\mathcal{A}$. For given $w \geq 1$, deciding if a given structure has treewidth $\leq w$ and, if so, to compute a tree decomposition of width w, is FPT [4]. We often have to assume that the elements in a bag $\chi(n)$ are ordered (in an arbitrary way). In this case, $\chi(n)$ is a tuple of elements, denoted as $\boldsymbol{a}$. By slight abuse of notation, we shall nevertheless apply set operations to such tuples (e.g., $a \in \boldsymbol{a}$, $\boldsymbol{a} \cap \boldsymbol{b}$) with the obvious meaning.

A tree decomposition $\mathcal{T} = (T, \chi)$ is called *normalized* (or *nice*) [19] if: (1) Each $n \in T$ has at most two children. (2) For each $n \in T$ with two children n_1, n_2, $\chi(n) = \chi(n_1) = \chi(n_2)$. (3) For each $n \in T$ with one child n', $\chi(n)$ and $\chi(n')$ differ in at most one element (i.e., $|\chi(n) \Delta \chi(n')| \leq 1$). (4) Leaf nodes $n \in T$ have empty bags (i.e., $\chi(n) = \emptyset$). W.l.o.g., we assume that every tree decomposition is normalized, since this normalization can be obtained in linear time without increasing the width [19].

A σ-structure $\mathcal{A}$ can be extended in order to additionally represent a tree decomposition $\mathcal{T}$ of $\mathcal{A}$. To this end, we extend σ to $\sigma_{td} = \sigma \cup \{\textit{root}, \textit{leaf}, \textit{child}_1, \textit{child}_2, \textit{bag}\}$ by unary relation symbols *root* and *leaf* with the obvious meaning and binary relation symbols *child*$_1$ and *child*$_2$. Thereby *child*$_1(n_1, n)$ denotes that n_1 is the first or the only child of n and *child*$_2(n_2, n)$ denotes that n_2 is the second child of n. Finally, *bag* has arity $w + 2$, where *bag*$(n, a_0, \ldots, a_w)$ expresses $\chi(n) = (a_0, \ldots, a_w)$. For bags of smaller size, we assume that the remaining positions in the *bag*-predicate are padded with dummy elements. We write $\mathcal{A}_{td}$ to denote the σ_{td}-structure representing both $\mathcal{A}$

and $\mathcal{T}$, i.e., the domain of $\mathcal{A}_{td}$ consists of $\mathrm{dom}(\mathcal{A})$ plus the nodes of $\mathcal{T}$. Moreover, the relations in $\mathcal{A}_{td}$ coincide with $\mathcal{A}$ for every $R \in \sigma$ and in addition, $\mathcal{A}_{td}$ contains a representation of $\mathcal{T}$ via appropriate relations for $R \in \{\mathit{root}, \mathit{leaf}, \mathit{child}_1, \mathit{child}_2, \mathit{bag}\}$.

The nodes of tree decompositions are either element introduction (EI), element removal (ER), permutation (P), branch (B), or leaf (L) nodes. Thereby "element introduction", "element removal", and "permutation" refers to the way in which the bag of some node is obtained from the bag of its child.

Monadic Second-Order Logic (MSO). MSO extends first-order logic by the use of *set variables* or *second-order variables* (denoted by upper case letters), which range over sets of domain elements. In contrast, the *individual variables* or *first-order variables* (denoted by lower case letters) range over single domain elements. The *quantifier depth* of an MSO-formula φ is defined as the maximum degree of nesting of quantifiers (both for individual and set variables) in φ.

Let $\varphi(\boldsymbol{x}, \boldsymbol{X})$ be an MSO-formula with free variables $\boldsymbol{x} = (x_1, \ldots, x_n)$ and $\boldsymbol{X} = (X_1, \ldots, X_m)$. Furthermore, let $\mathcal{A}$ be a σ-structure with $A = \mathrm{dom}(\mathcal{A})$, let $\boldsymbol{a} \in A^n$ and $\boldsymbol{A} \in \mathcal{P}(A)^m$, where $\mathcal{P}(A)$ denotes the powerset of A. We write $(\mathcal{A}, \boldsymbol{a}, \boldsymbol{A}) \models \varphi(\boldsymbol{x}, \boldsymbol{X})$ to denote that $\varphi(\boldsymbol{a}, \boldsymbol{A})$ evaluates to true in $\mathcal{A}$. Usually, we refer to $(\mathcal{A}, \boldsymbol{a}, \boldsymbol{A})$ simply as a "structure" rather than a "structure with distinguished elements and sets". We call $(\boldsymbol{a}, \boldsymbol{A})$ a model of φ over $\mathcal{A}$ and denote by $\varphi(\mathcal{A})$ the set of all models over $\mathcal{A}$.

We call structures $(\mathcal{A}, \boldsymbol{a}, \boldsymbol{A})$ and $(\mathcal{B}, \boldsymbol{b}, \boldsymbol{B})$ *k-equivalent* and write $(\mathcal{A}, \boldsymbol{a}, \boldsymbol{A}) \equiv_k^{MSO} (\mathcal{B}, \boldsymbol{b}, \boldsymbol{B})$, iff for every MSO-formula $\varphi(\boldsymbol{x}, \boldsymbol{X})$ of quantifier depth $\leq k$, the equivalence $(\mathcal{A}, \boldsymbol{a}, \boldsymbol{A}) \models \varphi(\boldsymbol{x}, \boldsymbol{X}) \Leftrightarrow (\mathcal{B}, \boldsymbol{b}, \boldsymbol{B}) \models \varphi(\boldsymbol{x}, \boldsymbol{X})$ holds. By definition, $\equiv_k^{MSO}$ is an equivalence relation possessing only finitely many equivalence classes for any k. These classes are referred to as *k-types* or simply as *types*. There is a nice characterization of k-equivalence by *Ehrenfeucht-Fraïssé games*: The k-round MSO-game on two structures $(\mathcal{A}, \boldsymbol{a}, \boldsymbol{A})$ and $(\mathcal{B}, \boldsymbol{b}, \boldsymbol{B})$ is played between two players – the spoiler and the duplicator. Thereby $(\mathcal{A}, \boldsymbol{a}, \boldsymbol{A})$ and $(\mathcal{B}, \boldsymbol{b}, \boldsymbol{B})$ are k-equivalent iff the duplicator has a winning strategy in the game on these structures. For details, see e.g. [21].

Datalog. We assume some familiarity with datalog, see e.g. [5]. Syntactically, a datalog program Π is a set of function-free, definite Horn clauses, i.e., each clause consists of a non-empty head and a possibly empty body. The (minimal-model) semantics can be defined as the least fixpoint (LFP) of applying the immediate consequence operator. A predicate is called *extensional* if it occurs only in the body of the rules in Π, whereas predicates also occurring in the heads are called *intensional*.

Let $\mathcal{A}$ be a σ-structure. In the context of datalog, it is convenient to think of the relations $R^{\mathcal{A}}$ with $R \in \sigma$ as sets of ground atoms. The set of all such ground atoms of a structure $\mathcal{A}$ is referred to as the extensional database (EDB) of $\mathcal{A}$, which we shall denote as $\mathcal{E}(\mathcal{A})$. We have $R(\boldsymbol{a}) \in \mathcal{E}(\mathcal{A})$ iff $\boldsymbol{a} \in R^{\mathcal{A}}$.

The fragment of *quasi-guarded datalog* has been recently introduced in [14]:

Definition 1. *Let σ_{td} be the extension of a signature σ and let Π be a datalog program over σ_{td}. Moreover, let r be a rule in Π and let x, y be variables in r. Then y is called* functionally dependent on x in one step*, if the body of r contains an atom* *of one of the following forms: child*$_1(x,y)$*, child*$_1(y,x)$*, child*$_2(x,y)$*, child*$_2(y,x)$*, or bag*$(x,a_0,\dots,a_k)$ *with* $y = a_i$ *for some* $i \in \{1,\dots,k\}$*.*

Consequently, y *is called* functionally dependent on x *if there exists some* $n \geq 1$ *and variables* $z_0,\dots,z_n$ *in* r *with* $z_0 = x, z_n = y$ *and, for every* $i \in \{1,\dots,n\}$*,* z_i *is functionally dependent on* z_{i-1} *in one step.*

Furthermore, Π *is called* quasi-guarded *if every rule* $r \in \Pi$ *contains an extensional atom* B*, s.t., every variable occurring in* r *either occurs in* B *or is functionally dependent on some variable in* B*.*

Evaluating a datalog program Π over a structure $\mathcal{A}$ is EXPTIME-complete [23] in general. It becomes NP-complete [10] if the arity of the intensional predicate symbols is bounded. Quasi-guarded datalog programs Π can be evaluated over σ_{td}-structures $\mathcal{A}_{td}$ in time $\mathcal{O}(|\Pi| * |\mathcal{A}_{td}|)$ [14].

We extend quasi-guarded datalog by adding counter variables. Such variables are integers which may be used as an additional argument for intensional predicates. An intensional predicate p having n arguments plus a counter is denoted by $p(t_1,\dots,t_n,j)$. Thereby the value of j is required to be fully determined by the grounding of $t_1,\dots,t_n$. If the predicate p occurs in the head of some rule r, then the counter j may have one of the following four forms: (1) j is initialized by a constant c (e.g., $p(t_1,\dots,t_n,c)$). (2) j takes the value of some j' being already present in the body of r (e.g., $p(t_1,\dots,t_n,j')$). (3) j is the product of two counters j_1, j_2 occurring in r's body (e.g., $p(t_1,\dots,t_n,j_1 * j_2)$). Additionally, we allow rules without being quasi-guarded but having the following strict form:

$$p(t_1,\dots,t_n,\mathrm{SUM}(j)) \leftarrow q(t_1,\dots,t_m,j).$$

where p, q are intensional predicates and $m > n$. In this case the semantics is similar to the SUM-aggregate function in ordinary SQL, where one first applies a GROUP BY over the variables $t_1,\dots,t_n$. Consider for example the predicates $\mathit{instLecture}(n,j)$ stating that there are j lectures held at institute n, as well as $\mathit{persLecture}(n,p,j)$ stating that there are j lectures held by lecturer p at institute n. A possible rule expressing the relationship between these two predicates would be

$$\mathit{instLecture}(n,\mathrm{SUM}(j)) \leftarrow \mathit{persLecture}(n,p,j).$$

whose meaning is also captured by the SQL query

```
SELECT n, SUM(j) FROM persLecture GROUP BY n.
```

We call the resulting fragment *quasi-guarded extended datalog*. For a formal definition of the semantics of SUM, see [18]. Aggregate functions are well studied, starting with the first formalizations of Klug [20] through to more recent work, e.g. [1, 6, 16].

3 Counting Problems

In this section, we consider MSO-definable counting problems. We start by giving a formal definition of a more general counting problem. Let C denote a class of structures and Φ a class of logical formulae. Then the counting variant of model checking (MC) is defined as follows:

#MC(C,Φ)
Instance: A structure $\mathcal{A} \in C$ and a formula $\varphi \in \Phi$.
Parameter: $\|\varphi\|$.
Problem: Compute $|\varphi(\mathcal{A})|$.

Thereby $\|\cdot\|$ denotes the size of a reasonable encoding. In this paper, we only consider the case $\Phi = \text{MSO}$, i.e., φ is an MSO-formula. Moreover, when we study the relationship between #MC(C,MSO) problems and datalog, C will be restricted to structures whose treewidth is bounded by some constant.

We illustrate the above concepts by expressing #3-COLORABILITY (i.e., the counting variant of 3-COLORABILITY) as a #MC(C,MSO)-problem. As the class C we have the set of finite, undirected graphs or, equivalently, the σ-structures where σ consists of a single, binary relation symbol $edge$. We can define an MSO-formula $\varphi(R, G, B)$ as follows, expressing that the sets R, G, B form a valid 3-coloring:

$$\varphi(R, G, B) \equiv \mathit{Partition}(R, G, B) \wedge \forall v_1 \forall v_2 [\mathit{edge}(v_1, v_2) \rightarrow (\neg R(v_1) \vee \neg R(v_2)) \wedge (\neg G(v_1) \vee \neg G(v_2)) \wedge (\neg B(v_1) \vee \neg B(v_2))],$$

where $\mathit{Partition}(R, G, B)$ is used as a short-hand for the following formula:

$$\mathit{Partition}(R, G, B) \equiv \forall v[(R(v) \vee G(v) \vee B(v)) \wedge (\neg R(v) \vee \neg G(v)) \wedge (\neg R(v) \vee \neg B(v)) \wedge (\neg G(v) \vee \neg B(v))].$$

Suppose that a graph (V, E) is given by an $\{\mathit{edge}\}$-structure $\mathcal{A}$ having domain $\mathrm{dom}(\mathcal{A}) = V$ and relation $\mathit{edge}^{\mathcal{A}} = E$. The above formula $\varphi(R, G, B)$ is one possible way of expressing that the sets R, G, B of vertices form a valid 3-coloring, i.e., R, G, B form a partition of V and, for every pair of vertices v_1, v_2, if they are adjacent then they are not both in R or both in G or both in B. Then we obtain the number of valid 3-colorings of (V, E) as

$$\left|\{\boldsymbol{A} \in \mathcal{P}(V)^3 \mid (\mathcal{A}, \boldsymbol{A}) \models \varphi(R, G, B)\}\right|,$$

i.e., the number of possible assignments to the free set-variables R, G, B in φ to make φ true in $\mathcal{A}$.

It is convenient to define *MSO-definable counting problems* as #MC(C,MSO) problems for formulae φ without free first-order variables. Note that this means no loss of generality since any free individual variable x can be replaced by a set variable X plus an appropriate conjunct in φ which guarantees that X is a singleton.

The primary goal of this section is to show that every MSO-definable counting problem over structures with bounded treewidth can be expressed by a program Π in the quasi-guarded extended datalog fragment defined in Section 2. Actually, *expressing* a problem in datalog also means *solving* the problem since, in contrast to MSO, datalog also has an operational semantics in addition to its declarative semantics. We shall therefore also analyze the complexity of evaluating such programs. This will ultimately allow us to give an alternative proof of the fixed-parameter linear time upper bound (assuming unit cost for arithmetic operations) on this class of counting problems.

The generic construction of a program $\mathit{\Pi}$ corresponding to an MSO-formula φ (which will be detailed in the proof of Theorem 1) crucially depends on traversing a tree decomposition $\mathcal{T}$ in a bottom-up manner and reasoning about the k-type of the structure induced by the subtree of $\mathcal{T}$ rooted at each node n. Lemma 1 below allows us to establish the connection between the k-type of the structure induced by the subtree rooted at n and the k-type of the structure(s) induced by the subtree(s) rooted at the only child (resp. the two children) of n. We first define some additional terminology, which will be helpful for the formulation of this lemma.

Definition 2. *Let T be a tree with a node $n \in T$. Then we denote the* subtree rooted at n *as T_n. Likewise, let $\mathcal{A}$ be a finite structure and let $\mathcal{T} = (T, \chi)$ be a tree decomposition of $\mathcal{A}$. Then we define $\mathcal{T}_n$ as the restriction of $\mathcal{T}$ to the nodes of T_n and we denote by $\mathcal{A}_n$ the substructure of $\mathcal{A}$ induced by the elements of the bags of $\mathcal{T}_n$.*

Definition 3. *Let $m \geq 1$ be an integer and let $\mathcal{A}$ and $\mathcal{B}$ be σ-structures. Moreover, let $\boldsymbol{a} = (a_0, \ldots, a_m)$ and $\boldsymbol{b} = (b_0, \ldots, b_m)$ be tuples with $a_i \in \mathrm{dom}(\mathcal{A})$ and $b_i \in \mathrm{dom}(\mathcal{B})$. We call $\boldsymbol{a}$ and $\boldsymbol{b}$* equivalent *and write $\boldsymbol{a} \equiv \boldsymbol{b}$, iff for all predicate symbols $R \in \sigma$ with $\alpha = \mathrm{arity}(R)$ and for all tuples $(i_1, \ldots, i_\alpha) \in \{0, \ldots, m\}^\alpha$, the equivalence $R^{\mathcal{A}}(a_{i_1}, \ldots, a_{i_\alpha}) \Leftrightarrow R^{\mathcal{B}}(b_{i_1}, \ldots, b_{i_\alpha})$ holds.*

Lemma 1. *Given σ-structures $\mathcal{A}$ and $\mathcal{B}$, let $\mathcal{S}$ (resp. $\mathcal{T}$) be a normalized tree decomposition of $\mathcal{A}$ (resp. $\mathcal{B}$) having width w and let n (resp. m) be an internal node in $\mathcal{S}$ (resp. $\mathcal{T}$). Let n' (resp. m') denote the only or left child of n (resp. m) and let n'' (resp. m'') denote the optional right child. Let $\boldsymbol{a}, \boldsymbol{a}', \boldsymbol{a}'', \boldsymbol{b}, \boldsymbol{b}'$, and $\boldsymbol{b}''$ denote the bags of nodes n, n', n'', m, m', and m'', respectively. Furthermore consider l-tuples of domain-subsets $\boldsymbol{X} \in \mathcal{P}(\mathrm{dom}(\mathcal{A}_n))^l$, $\boldsymbol{X}' \in \mathcal{P}(\mathrm{dom}(\mathcal{A}_{n'}))^l$, $\boldsymbol{X}'' \in \mathcal{P}(\mathrm{dom}(\mathcal{A}_{n''}))^l$, $\boldsymbol{Y} \in \mathcal{P}(\mathrm{dom}(\mathcal{B}_m))^l$, $\boldsymbol{Y}' \in \mathcal{P}(\mathrm{dom}(\mathcal{B}_{m'}))^l$ and $\boldsymbol{Y}'' \in \mathcal{P}(\mathrm{dom}(\mathcal{B}_{m''}))^l$.*

(P) nodes: *Let n and m be of type (P). If $\boldsymbol{X} = \boldsymbol{X}'$, $\boldsymbol{Y} = \boldsymbol{Y}'$ and there exists a permutation π, s.t. $\boldsymbol{a} = \pi(\boldsymbol{a}')$ and $\boldsymbol{b} = \pi(\boldsymbol{b}')$, then $(\mathcal{A}_{n'}, \boldsymbol{a}', \boldsymbol{X}') \equiv_k^{MSO} (\mathcal{B}_{m'}, \boldsymbol{b}', \boldsymbol{Y}')$ implies $(\mathcal{A}_n, \boldsymbol{a}, \boldsymbol{X}) \equiv_k^{MSO} (\mathcal{B}_m, \boldsymbol{b}, \boldsymbol{Y})$.*

(ER) nodes: *Let n and m be of type (ER), s.t. $\boldsymbol{a}' \setminus \boldsymbol{a} = \{a_j\}$ and $\boldsymbol{b}' \setminus \boldsymbol{b} = \{b_j\}$, i.e. the removed elements have the same index. If $\boldsymbol{X} = \boldsymbol{X}'$ and $\boldsymbol{Y} = \boldsymbol{Y}'$, then $(\mathcal{A}_{n'}, \boldsymbol{a}', \boldsymbol{X}') \equiv_k^{MSO} (\mathcal{B}_{m'}, \boldsymbol{b}', \boldsymbol{Y}')$ implies $(\mathcal{A}_n, \boldsymbol{a}, \boldsymbol{X}) \equiv_k^{MSO} (\mathcal{B}_m, \boldsymbol{b}, \boldsymbol{Y})$.*

(EI) nodes: *Let n and m be of type (EI), s.t. $\boldsymbol{a} \setminus \boldsymbol{a}' = \{a_j\}$ and $\boldsymbol{b} \setminus \boldsymbol{b}' = \{b_j\}$. If $\boldsymbol{a} \equiv \boldsymbol{b}$ and there exists $(\varepsilon_1, \ldots, \varepsilon_l) \in \{0,1\}^l$, s.t. $X_i = X_i'$ respectively $Y_i = Y_i'$ if $\varepsilon_i = 0$, and $X_i = X_i' \cup \{a_j\}$ respectively $Y_i = Y_i' \cup \{b_j\}$ if $\varepsilon_i = 1$, then $(\mathcal{A}_{n'}, \boldsymbol{a}', \boldsymbol{X}') \equiv_k^{MSO} (\mathcal{B}_{m'}, \boldsymbol{b}', \boldsymbol{Y}')$ implies $(\mathcal{A}_n, \boldsymbol{a}, \boldsymbol{X}) \equiv_k^{MSO} (\mathcal{B}_m, \boldsymbol{b}, \boldsymbol{Y})$.*

(B) nodes: *Let n and m be of type (B). If $\boldsymbol{X} = (X_1' \cup X_1'', \ldots, X_n' \cup X_n'')$ and $\boldsymbol{Y} = (Y_1' \cup Y_1'', \ldots, Y_n' \cup Y_n'')$, then $(\mathcal{A}_{n'}, \boldsymbol{a}', \boldsymbol{X}') \equiv_k^{MSO} (\mathcal{B}_{m'}, \boldsymbol{b}', \boldsymbol{Y}')$ and $(\mathcal{A}_{n''}, \boldsymbol{a}'', \boldsymbol{X}'') \equiv_k^{MSO} (\mathcal{B}_{m''}, \boldsymbol{b}'', \boldsymbol{Y}'')$ imply $(\mathcal{A}_n, \boldsymbol{a}, \boldsymbol{X}) \equiv_k^{MSO} (\mathcal{B}_m, \boldsymbol{b}, \boldsymbol{Y})$.*

Proof Idea. The proof proceeds by a case distinction over the four possible types of internal nodes n and m in a normalized tree decomposition. All cases are shown by an easy argument using Ehrenfeucht-Fraïssé games (see [21]). By the k-equivalence $(\mathcal{A}_{n'}, \boldsymbol{a}', \boldsymbol{X}') \equiv_k^{MSO} (\mathcal{B}_{m'}, \boldsymbol{b}', \boldsymbol{Y}')$ and, optionally, $(\mathcal{A}_{n''}, \boldsymbol{a}'', \boldsymbol{X}'') \equiv_k^{MSO} (\mathcal{B}_{m''}, \boldsymbol{b}'', \boldsymbol{Y}'')$, the duplicator has a winning strategy in the k-round game played on the structures $(\mathcal{A}_{n'}, \boldsymbol{a}', \boldsymbol{X}')$ and $(\mathcal{B}_{m'}, \boldsymbol{b}', \boldsymbol{Y}')$ as well as on the structures $(\mathcal{A}_{n''}, \boldsymbol{a}'', \boldsymbol{X}'')$ and

$(\mathcal{B}_{m''}, \boldsymbol{b}'', \boldsymbol{Y}'')$. Then the winning strategy at the only child node of n and m (resp. at the two child nodes of n and m) can be extended (resp. combined) to a winning strategy in the game played on the structures $(\mathcal{A}_n, \boldsymbol{a}, \boldsymbol{X})$ and $(\mathcal{B}_m, \boldsymbol{b}, \boldsymbol{Y})$. Actually, in case of (P) and (ER) nodes, the winning strategy for $(\mathcal{A}_{n'}, \boldsymbol{a}', \boldsymbol{X}')$ and $(\mathcal{B}_{m'}, \boldsymbol{b}', \boldsymbol{Y}')$ may even be left unchanged. In order to extend the winning strategy in case of an (EI)-node and to combine the winning strategies in case of a (B)-node, the connectedness condition of tree decompositions is crucial. □

Lemma 1 gives us the intuition how to determine the k-type of the substructure induced by a subtree $\mathcal{T}_n$ via a bottom-up traversal of the tree decomposition $\mathcal{T}$. Essentially, the type of the structure induced by $\mathcal{T}_n$ is fully determined by three components: (i) the type of the structure induced by the subtree rooted at the child node(s) of n, (ii) the relations between elements in the bag at node n, and (iii) the intersection of the distinguished domain elements $\boldsymbol{a}$ and distinguished sets $\boldsymbol{X}$ with the elements in the bag at node n.

We now have a closer look at the distinguished sets and their effect on the type of a structure. Suppose that we have fixed some structure $\mathcal{A}$ together with distinguished elements $\boldsymbol{a}$. Then the question is if different choices $\boldsymbol{A}$ and $\boldsymbol{B}$ of distinguished sets necessarily lead to different types. In the following lemma we give a positive answer to this question for the case that $\boldsymbol{A}$ and $\boldsymbol{B}$ differ on an element in $\boldsymbol{a}$. This lemma will be very useful when, on our bottom-up traversal of the tree decomposition, we encounter an element introduction node: Let a denote the element that is new w.r.t. the bag at the child node and suppose that we are considering l distinguished sets. Then, by the lemma below, we know that each of the 2^l possible choices of either adding the element a to each of the l distinguished sets or not necessarily produces a different type. This property in turn is important for solving the #MC(C,MSO) problem since it guarantees that we do not count any solution twice. Similarly, in Section 4, it will keep us from outputting a solution twice.

Lemma 2. *Given a σ-structure $\mathcal{A}$ with $\boldsymbol{a} \in \mathrm{dom}(\mathcal{A})^m$ and $\boldsymbol{A}, \boldsymbol{B} \in \mathcal{P}(\mathrm{dom}(\mathcal{A}))^l$. If there exists an index $i \in \{1, \dots, m\}$, s.t. $A_i \cap \boldsymbol{a} \neq B_i \cap \boldsymbol{a}$, then it follows that $(\mathcal{A}, \boldsymbol{a}, \boldsymbol{A}) \not\equiv_k^{MSO} (\mathcal{A}, \boldsymbol{a}, \boldsymbol{B})$.*

Proof. This follows directly from the definition of $\equiv_k^{MSO}$ through Ehrenfeucht-Fraïssé games. Indeed, suppose that some domain element $a \in \boldsymbol{a}$ is contained in some A_i but not in B_i (or vice versa). Then the spoiler can win the game on the structures $(\mathcal{A}, \boldsymbol{a}, \boldsymbol{A})$ and $(\mathcal{A}, \boldsymbol{a}, \boldsymbol{B})$ in a single move, simply by choosing a.

We introduce one more auxiliary definition and then we will be ready to formulate and prove the main result of this section, namely Theorem 1.

Definition 4. *Let $\mathcal{A}$ be a σ-structure and let $\boldsymbol{a}$ be a tuple of elements of* $\mathrm{dom}(\mathcal{A})$. *Then $\mathcal{R}(\boldsymbol{a})$ denotes the set of all* ground atoms *with predicates in σ and arguments in $\boldsymbol{a}$, i.e. $\mathcal{R}(\boldsymbol{a}) = \{R(a_1, \dots, a_\alpha) \mid R \in \sigma, \alpha = \mathrm{arity}(R), a_1, \dots, a_\alpha \in \boldsymbol{a}\}$.*

Theorem 1. *Let signature σ and integer $w \geq 1$ be arbitrary but fixed. For the class C of σ-structures of treewidth at most w, the problem* #MC(C,MSO) *is definable in the quasi-guarded fragment of extended datalog over σ_{td}.*

Proof. Let $\varphi(\boldsymbol{X})$ be an arbitrary MSO-formula with free second-order variables $\boldsymbol{X} = X_1, \ldots, X_l$ and quantifier depth k. We will construct a quasi-guarded extended datalog program $\mathit{\Pi}$ with a distinguished predicate *solution*, s.t. for any σ-structure $\mathcal{C} \in C$, *solution*(j) is in the LFP of $\mathit{\Pi} \cup \mathcal{C}_{td}$ iff $|\varphi(\mathcal{C})| = j$.

During the construction we maintain a set $\mathit{\Theta}$ of rank-k types ϑ of structures of the form $(\mathcal{A}, \boldsymbol{a}, \boldsymbol{A})$, where $\mathrm{tw}(\mathcal{A}) \leq w$, $\mathcal{T}$ is a tree decomposition of $\mathcal{A}$ with width $\mathrm{tw}(\mathcal{A})$, $n = \mathrm{root}(\mathcal{T})$, $\boldsymbol{a} = \mathrm{bag}(n)$ and $\boldsymbol{A} \in \mathcal{P}(\mathrm{dom}(\mathcal{A}))^l$. For each type $\vartheta \in \mathit{\Theta}$, we save a witness denoted by $W(\vartheta) = (\mathcal{A}, \boldsymbol{a}, \boldsymbol{A})$. The types $\vartheta \in \mathit{\Theta}$ will serve as names of binary predicates in our program $\mathcal{P}$ as well as names for constants. No confusion will arise from this "overloading" since the meaning will always be clear from the context.

It is important to notice that the construction of program $\mathit{\Pi}$ only depends on $\varphi(\boldsymbol{X})$ and the upper bound w on the treewidth of the structures in C but not on a concrete structure $\mathcal{C} \in C$. But of course, $\mathit{\Pi}$ will ultimately be used to compute $|\varphi(\mathcal{C})|$ for a concrete input structure $\mathcal{C}$. The intended meaning of the ϑ-predicate (which holds for any structure $\mathcal{C} \in C$) in our program construction is captured by Property A below.

Let $\vartheta \in \mathit{\Theta}$, let $\mathcal{C} \in C$ and let n be a node in a tree decomposition $\mathcal{T}$ of $\mathcal{C}$, s.t. the bag at n is $\boldsymbol{b}$. Then we define the set $\mathit{\Gamma}(n, \vartheta)$ as

$$\mathit{\Gamma}(n, \vartheta) = \{\boldsymbol{B} \in \mathcal{P}(\mathrm{dom}(\mathcal{C}_n))^l \mid (\mathcal{C}_n, \boldsymbol{b}, \boldsymbol{B}) \equiv_k^{MSO} W(\vartheta)\}.$$

(Recall from Definition 2 that $\mathcal{C}_n$ denotes the substructure of $\mathcal{C}$ induced by the elements in the bags of the subtree $\mathcal{T}_n$ rooted at n.) Then, we have

Property A. For every $j \geq 1$, there exists an atom $\vartheta(n, j)$ in the LFP of $\mathit{\Pi} \cup \mathcal{C}_{td}$ iff $|\mathit{\Gamma}(n, \vartheta)| = j$. Furthermore, there exists no atom $\vartheta(n, \_)$ in the LFP, iff $|\mathit{\Gamma}(n, \vartheta)| = 0$.

We shall show that Property A indeed holds at the end of the proof. First, we give the details of the construction of program $\mathit{\Pi}$. Initially, we set $\mathit{\Theta} = \mathit{\Pi} = \emptyset$. Then we construct $\mathit{\Theta}$ by structural induction over normalized tree decompositions. Since there are only finitely many MSO rank-k types for structures $(\mathcal{A}, \boldsymbol{a}, \boldsymbol{A})$ [21], the induction will eventually halt. For the base case of the induction, we consider a tree decomposition $\mathcal{T}$ consisting of a single node n with empty bag $\boldsymbol{a}$. To create structure $(\mathcal{A}, \boldsymbol{a}, \boldsymbol{A})$ having tree decomposition $\mathcal{T}$, let $\mathcal{A}$ be the σ-structure with $\mathrm{dom}(\mathcal{A}) = \emptyset$. Moreover $\boldsymbol{A} = (\emptyset, \ldots, \emptyset)$. Now we invent a new token ϑ, add it to $\mathit{\Theta}$ and save the witness $W(\vartheta) = (\mathcal{A}, \boldsymbol{a}, \boldsymbol{A})$. Additionally we add the following rule to $\mathit{\Pi}$:

$$\vartheta(n, 1) \leftarrow \mathit{leaf}(n).$$

In the induction step, we construct all possible structures $(\mathcal{A}, \boldsymbol{a}, \boldsymbol{A})$ that can be created by extending the tree decomposition of witness $W(\vartheta') = (\mathcal{A}', \boldsymbol{a}', \boldsymbol{A}')$ for any $\vartheta' \in \mathit{\Theta}$ in a "bottom-up" manner by introducing a new root node. Let $\boldsymbol{a}' = (a_0, \ldots, a_m)$ be the bag of the old root n'. The new root n can be of any of the following node types:

(P) node: Consider all possible permutations π of the indices $\{0, \ldots, m\}$ and set $\boldsymbol{a} = (a_{\pi(0)}, \ldots, a_{\pi(m)})$, $\mathcal{A} = \mathcal{A}'$, and $\boldsymbol{A} = \boldsymbol{A}'$. For each of these $(\mathcal{A}, \boldsymbol{a}, \boldsymbol{A})$, we check whether there exists $\vartheta \in \mathit{\Theta}$ with witness $W(\vartheta) = (\mathcal{B}, \boldsymbol{b}, \boldsymbol{B})$, s.t. $(\mathcal{A}, \boldsymbol{a}, \boldsymbol{A}) \equiv_k^{MSO} (\mathcal{B}, \boldsymbol{b}, \boldsymbol{B})$. If such a ϑ is found we take it, otherwise we invent a new token ϑ, add it to $\mathit{\Theta}$ and save the witness $W(\vartheta) = (\mathcal{A}, \boldsymbol{a}, \boldsymbol{A})$. In either case, we add the following rules to Π. Note that we do not write down the dummy elements for smaller bags.

$$\begin{aligned} \mathit{aux}_P(n, \vartheta, \vartheta', j) &\leftarrow \mathit{bag}(n, x_{\pi(0)}, \dots, x_{\pi(m)}), \mathit{child}(n', n), \\ &\qquad \mathit{bag}(n', x_0, \dots, x_m), \vartheta'(n', j). \\ \vartheta(n, j) &\leftarrow \mathit{aux}_P(n, \vartheta, \_, j). \end{aligned}$$

(ER) node: Set $\boldsymbol{a} = (a_1, \dots, a_m)$, $\mathcal{A} = \mathcal{A}'$ and $\boldsymbol{A} = \boldsymbol{A}'$. For each of these $(\mathcal{A}, \boldsymbol{a}, \boldsymbol{A})$ we check whether there exists $\vartheta \in \Theta$ with $W(\vartheta) = (\mathcal{B}, \boldsymbol{b}, \boldsymbol{B})$, s.t. $(\mathcal{A}, \boldsymbol{a}, \boldsymbol{A}) \equiv_k^{MSO} (\mathcal{B}, \boldsymbol{b}, \boldsymbol{B})$. If no such ϑ is found, we invent a new token ϑ, add it to Θ and save $W(\vartheta) = (\mathcal{A}, \boldsymbol{a}, \boldsymbol{A})$. In either case, the following rule is added to Π:

$$\begin{aligned} \mathit{aux}_R(n, \vartheta, \vartheta', j) &\leftarrow \mathit{bag}(n, x_1, \dots, x_m), \mathit{child}_1(n', n), \\ &\qquad \mathit{bag}(n', x_0, x_1, \dots, x_m), \vartheta'(n', j). \end{aligned}$$

For all $\vartheta \in \Theta$ for which we created the rule above, we also add:

$$\vartheta(n, \mathrm{SUM}(j)) \leftarrow \mathit{aux}_R(n, \vartheta, \_, j).$$

(EI) node: Adding an (EI) node is only possible if $m < w$. We take a new element $a_{m+1} \notin \mathrm{dom}(\mathcal{A})$ and let $\boldsymbol{a} = (a_0, \dots, a_m, a_{m+1})$. All possible structures $\mathcal{A}$ can be generated by setting $\mathrm{dom}(\mathcal{A}) = \mathrm{dom}(\mathcal{A}') \cup \{a_{m+1}\}$ and by extending the EDB $\mathcal{E}(\mathcal{A}')$ to $\mathcal{E}(\mathcal{A})$ in the following way. Let $\Delta = \mathcal{E}(\mathcal{A}) \setminus \mathcal{E}(\mathcal{A}')$ be an arbitrary set of tuples, s.t. $\Delta \subseteq \mathcal{R}(\boldsymbol{a})$ and a_{m+1} occurs as an argument of all tuples in Δ. Furthermore we consider all possible tuples $\boldsymbol{\varepsilon} = (\varepsilon_1, \dots, \varepsilon_l) \in \{0, 1\}^l$ and extend $\boldsymbol{A}'$ to $\boldsymbol{A}$, s.t. $A_i = A_i'$ if $\varepsilon_i = 0$ and $A_i = A_i' \cup \{a_{m+1}\}$ if $\varepsilon_i = 1$. For every such structure, i.e. for each combination of $\mathcal{A}$ and $\boldsymbol{A}$, we check whether there exists $\vartheta \in \Theta$ with $W(\vartheta) = (\mathcal{B}, \boldsymbol{b}, \boldsymbol{B})$, s.t. $(\mathcal{A}, \boldsymbol{a}, \boldsymbol{A}) \equiv_k^{MSO} (\mathcal{B}, \boldsymbol{b}, \boldsymbol{B})$. If no such ϑ is found, we invent a new token ϑ, add it to Θ and save $W(\vartheta) = (\mathcal{A}, \boldsymbol{a}, \boldsymbol{A})$. In either case, the following rule is added to Π:

$$\begin{aligned} \mathit{aux}_I(n, \vartheta, \vartheta', \boldsymbol{\varepsilon}, j) &\leftarrow \mathit{bag}(n, x_0, \dots, x_{m+1}), \mathit{child}_1(n', n), \\ &\qquad \mathit{bag}(n', x_0, \dots, x_m), \vartheta'(n', j), \\ &\qquad \{R(x_{i_1}, \dots, x_{i_r}) \mid R(a_{i_1}, \dots, a_{i_r}) \in \mathcal{E}(\mathcal{A})\}, \\ &\qquad \{\neg R(x_{i_1}, \dots, x_{i_r}) \mid R(a_{i_1}, \dots, a_{i_r}) \notin \mathcal{E}(\mathcal{A})\}. \end{aligned}$$

For all $\vartheta \in \Theta$ for which we created the rule above, we also add:

$$\vartheta(n, \mathrm{SUM}(j)) \leftarrow \mathit{aux}_I(n, \vartheta, \_, \_, j).$$

(B) node: Let $\vartheta'' \in \Theta$ be a type with $W(\vartheta'') = (\mathcal{A}'', \boldsymbol{a}'', \boldsymbol{A}'')$, not necessarily distinct from ϑ'. W.l.o.g. we require $\boldsymbol{a}' = \boldsymbol{a}''$ and $\mathrm{dom}(\mathcal{A}') \cap \mathrm{dom}(\mathcal{A}'') = \boldsymbol{a}'$. Note that this can easily be achieved by renaming the elements of one of the two structures. Furthermore we check for inconsistency of the EDBs of the two structures $(\mathcal{A}', \boldsymbol{a}', \boldsymbol{A}')$ and $(\mathcal{A}'', \boldsymbol{a}', \boldsymbol{A}'')$, i.e. we check whether $\mathcal{E}(\mathcal{A}') \cap \mathcal{R}(\boldsymbol{a}') \neq \mathcal{E}(\mathcal{A}'') \cap \mathcal{R}(\boldsymbol{a}')$ or $\boldsymbol{A}' \cap \boldsymbol{a}' \neq \boldsymbol{A}'' \cap \boldsymbol{a}'$. If this is true, we ignore the pair, otherwise we create the new structure $(\mathcal{A}, \boldsymbol{a}, \boldsymbol{A})$ by setting $\mathrm{dom}(\mathcal{A}) = \mathrm{dom}(\mathcal{A}') \cup \mathrm{dom}(\mathcal{A}'')$, $\mathcal{E}(\mathcal{A}) = \mathcal{E}(\mathcal{A}') \cup \mathcal{E}(\mathcal{A}'')$, $\boldsymbol{a} = \boldsymbol{a}'$ and $A_i = A_i' \cup A_i''$. For every such structure, we check whether there exists $\vartheta \in \Theta$ with witness $W(\vartheta) = (\mathcal{B}, \boldsymbol{b}, \boldsymbol{B})$, s.t. $(\mathcal{A}, \boldsymbol{a}, \boldsymbol{A}) \equiv_k^{MSO} (\mathcal{B}, \boldsymbol{b}, \boldsymbol{B})$. If such a ϑ is found we take it, otherwise we invent a new token ϑ, add it to Θ and save the witness $W(\vartheta) = (\mathcal{A}, \boldsymbol{a}, \boldsymbol{A})$. In either case, we add the following rule to Π:

$$\begin{aligned} \mathit{aux}_B(n, \vartheta, \vartheta_1, \vartheta_2, j_1 * j_2) \leftarrow\ & \mathit{child}_1(n_1, n), \mathit{bag}(n_1, x_0, \ldots, x_m), \\ & \mathit{child}_2(n_2, n), \mathit{bag}(n_2, x_0, \ldots, x_m), \\ & \mathit{bag}(n, x_0, \ldots, x_m), \vartheta_1(n_1, j_1), \vartheta_2(n_2, j_2). \end{aligned}$$

For all $\vartheta \in \Theta$ for which we created the rule above, we also add:

$$\vartheta(n, \mathrm{SUM}(j)) \leftarrow \mathit{aux}_B(n, \vartheta, \_, \_, j).$$

Now that we have constructed Θ, the only thing left to do is the actual counting of solutions. To this end, we check for every $\vartheta \in \Theta$ with $W(\vartheta) = (\mathcal{A}, \boldsymbol{a}, \boldsymbol{A})$, whether $\mathcal{A} \models \varphi(\boldsymbol{A})$ holds and, if the answer is affirmative, we add the following rule to $\mathcal{P}$:

$$\mathit{aux}_{\mathit{root}}(\vartheta, j) \leftarrow \mathit{root}(n), \vartheta(n, j).$$

Finally we also add:

$$\mathit{solution}(\mathrm{SUM}(j)) \leftarrow \mathit{aux}_{\mathit{root}}(\_, j).$$

The bottom-up construction of Θ guarantees that we construct all possible rank-k types of structures $(\mathcal{A}, \boldsymbol{a}, \boldsymbol{A})$, where $\mathrm{tw}(\mathcal{A}) \leq w$ and $\boldsymbol{a}$ is the bag of the root node of a tree decomposition of $\mathcal{A}$. This follows from Lemma 1 and an easy induction argument.

Now consider an arbitrary input structure $\mathcal{C} \in C$ with tree decomposition $\mathcal{T}$. We have to show that *solution*(j) is in the LFP of $\Pi \cup \mathcal{C}_{td}$ iff $|\varphi(\mathcal{C})| = j$. By the above two rules with head predicates *aux*$_{\mathit{root}}$ and *solution*, it suffices to show that the ϑ-predicate has the desired Property A. We prove this by discussing the program rules added for each node type. Below, we restrict ourselves to the case that $|\Gamma(n, \vartheta)| \geq 1$. The case $|\Gamma(n, \vartheta)| = 0$ (i.e., no fact $\vartheta(n, \_)$ is in the LFP of $\Pi \cup \mathcal{C}_{td}$) is obvious.
(L) nodes: For an (L) node n with type ϑ, $\Gamma(n, \vartheta) = \{(\emptyset, \ldots, \emptyset)\}$. Since the only rule for deriving $\vartheta(n, j)$ sets $j = 1$, the statement holds.
(P) nodes: Rules for a (P) node n with type ϑ do not alter the counter j. But neither does a permutation of the elements in the root bag change the corresponding set $\Gamma(n, \vartheta)$.
(ER) nodes: For an (ER) node n with type ϑ, the rule deriving *aux*$_R$ does not alter j. If the type ϑ' of child n' changes for solution $\boldsymbol{A'}$ stored in $W(\vartheta')$ to the type ϑ, then by Lemma 1, this is also true for all $\boldsymbol{B'} \in \Gamma(n', \vartheta')$. For the second rule note that, for two types ϑ' and ϑ'' both leading to ϑ, we have $\Gamma(n', \vartheta') \cap \Gamma(n', \vartheta'') = \emptyset$. Thus $|\Gamma(n, \vartheta)|$ can be computed by summation over all the counters appearing in a derived fact *aux*$_R$.
(EI) nodes: For an (EI) node n with type ϑ, the rule deriving *aux*$_I$ does not alter j either. If the modification of solution $\boldsymbol{A'}$ stored in $W(\vartheta')$ according to the tuple $\boldsymbol{\varepsilon} = (\varepsilon_1, \ldots, \varepsilon_l)$ changes the type ϑ' of child n' to the type ϑ, then by Lemma 1, this is also true for all $\boldsymbol{B'} \in \Gamma(v', \vartheta')$ under the same modification $\boldsymbol{\varepsilon}$. Lemma 2 states that a different modification $\boldsymbol{\varepsilon'}$ leads to a type different from ϑ. Therefore and by the same argument as for (ER) nodes, $|\Gamma(n, \vartheta)|$ can be computed by summation over all the counters appearing in a derived fact *aux*$_I$.
(B) nodes: For a (B) node n, we know by Lemma 1, if the combination of solution $\boldsymbol{A_1}$ in $W(\vartheta_1)$ with solution $\boldsymbol{A_2}$ in $W(\vartheta_2)$ leads to ϑ, then any solution $\boldsymbol{B_1} \in \Gamma(n_1, \vartheta_1)$

combined with any solution $\boldsymbol{B_2} \in \Gamma(n_2, \vartheta_2)$ leads to ϑ. Since $\boldsymbol{B_1} \cap \boldsymbol{B_2} = \emptyset$ and from the connectedness condition of tree decompositions, it follows that there are indeed $|\Gamma(n_1, \vartheta_1)| * |\Gamma(n_2, \vartheta_2)| = j_1 * j_2$ possible different solutions that lead to ϑ through combination of ϑ_1 and ϑ_2. The second rule groups all solutions leading to ϑ together. From the discussion above and from the rules for *aux*$_{root}$ and *solution*, it follows that Π is the desired quasi-guarded extended datalog program.

So far, we have shown hat every MSO-definable counting problem is indeed solvable by quasi-guarded extended datalog. The following result complements this expressibility result by showing that quasi-guarded extended datalog programs can be evaluated in linear time.

Theorem 2. *Let Π be a quasi-guarded extended datalog program and let $\mathcal{A}$ be a finite structure. Then Π can be evaluated over $\mathcal{A}$ in time $\mathcal{O}(\|\Pi\| * \|\mathcal{A}\|)$, assuming unit cost for arithmetic operations. Thereby $\|\Pi\|$ denotes the size of the extended datalog program and $\|\mathcal{A}\|$ denotes the size of the data.*

Proof. Gottlob et al. [14] showed that this theorem holds for quasi-guarded datalog without the extension by counter variables. By definition, the counter is fully determined from the other arguments of the predicate. Therefore, the number of possible groundings of a rule r does not increase and hence the argument in [14] is still valid for quasi-guarded extended datalog without the $\mathrm{SUM}(\cdot)$ operator. On the other hand, it is easy to see, that adding rules with this operator does not harm the linear time bound, since we could replace rule r by rule r', where $\mathrm{SUM}(j)$ is substituted by j. The facts for r can then be derived by summation of the counter of r'. Since we assume unit cost for arithmetic operations, this does not violate the linear time bound.

Note that the construction and, therefore, the size of the datalog program Π in the proof of Theorem 1 only depends on the length of the formula φ and the treewidth w, but not on the structure $\mathcal{A}$. Hence, combining Theorem 1 and 2 immediately yields an alternative proof of the counting variant of Courcelle's Theorem as a corollary. Originally, this result was shown in [2] via the correspondence of MSO with FTAs.

Corollary 1. *Let C be the class of structures whose treewidth is bounded by some constant w. Then* #MC(C,MSO) *is solvable in fixed-parameter linear time, i.e. for each structure $\mathcal{A} \in C$ and formula $\varphi \in$ MSO it is solvable in time $\mathcal{O}(f(\|\varphi\|, w) \cdot \|\mathcal{A}\|)$.*

4 Enumeration Problems

We now show how the extended datalog programs from Theorem 1 can be used to solve also MSO-definable enumeration problems. Similarly as in Section 3, we start with a formal definition of the enumeration variant of model checking (MC).

#ENUM-MC(C,Φ)
Instance: A structure $\mathcal{A} \in C$ and a formula $\varphi \in \Phi$.
Parameter: $\|\varphi\|$.
Problem: Compute $\varphi(\mathcal{A})$.

As before, $\varphi(\mathcal{A})$ denotes the set of possible assignments to the free variables in φ that make φ true in $\mathcal{A}$. Again, we only consider the case that $\Phi = \text{MSO}$ and C is a class of structures with bounded treewidth. By *MSO-definable enumeration problems* we mean #ENUM-MC(C,MSO) problems for formulae φ without free first-order variables.

Let $\varphi(\boldsymbol{X})$ be an MSO-formula, Π be the extended datalog program from Theorem 1 to solve the counting problem defined by $\varphi(\boldsymbol{X})$, $\mathcal{A}$ be a σ-structure, and $\mathcal{A}_{td}$ denote the σ_{td}-structure representing both $\mathcal{A}$ and a tree decomposition $\mathcal{T}$ of $\mathcal{A}$. We cannot expect to use Π directly to compute the solutions of $\varphi(\boldsymbol{X})$. This would mean computing tuples of sets, which clearly surpasses the expressive power of datalog. However, we can use the facts in the LFP of $\Pi \cup \mathcal{A}_{td}$ (i.e., the intermediate results of our counting algorithm) and generate all solutions of $\varphi(\boldsymbol{X})$ in a postprocessing step. In Figure 1, we present the program enumerateSolutions, which is designed for precisely this purpose.

Our enumeration algorithm thus starts at the root node of $\mathcal{T}$ rather than at the leaves. Note that this is exactly what we have to do, since the root is the only node where types ϑ which correspond to some solutions of $\varphi(\boldsymbol{X})$ are identified. More precisely, our algorithm iterates over all types ϑ, s.t. the LFP of $\Pi \cup \mathcal{A}_{td}$ contains a fact $aux_{root}(\vartheta, j)$. By construction, these types correspond to the solutions of $\varphi(\boldsymbol{X})$. More formally, let $\boldsymbol{a}$ be the tuple of elements in the bag at the root of $\mathcal{T}$ and let $\boldsymbol{A}$ be a tuple of sets. Then the k-type of $(\mathcal{A}, \boldsymbol{a}, \boldsymbol{A})$ is some ϑ, s.t. $aux_{root}(\vartheta, \_)$ is in the LFP of $\Pi \cup \mathcal{A}_{td}$ iff $\mathcal{A} \models \varphi(\boldsymbol{A})$.

We can then exploit further facts from the LFP of $\Pi \cup \mathcal{A}_{td}$ to construct the actual solutions. Hereby, we collect for all nodes in $\mathcal{T}$ the types which *contributed* to the currently processed ϑ. We do so by constructing a new tree Υ for each such ϑ, which is stored in an internal data structure and described in detail below. Function getSol$(\cdot)$ in Figure 1 then traverses Υ several times until all solutions $\boldsymbol{A}$ of $\varphi(\boldsymbol{X})$ corresponding to ϑ are found. Below, we give a more detailed description of this procedure and all the auxiliary procedures used inside it.

Let us first describe the construction of the tree Υ for a type ϑ. This is done in the auxiliary procedure initTree(ϑ) of enumerateSolutions. In fact, Υ has as root node (n, ϑ), where n is the root node of tree decomposition $\mathcal{T}$. We then recursively add to a node (n', ϑ') in Υ a child node (n'', ϑ''), if $child_1(n'', n')$ and $aux_\lambda(n', \vartheta', \vartheta'', \ldots)$ (for $\lambda \in \{P, R, I\}$) are in the LFP of $\Pi \cup \mathcal{A}_{td}$. In case n' is a (B) node, we analogously add nodes using $aux_B(n', \vartheta', \vartheta_1, \vartheta_2, \_)$ but distinguish between left and right children. We store the children of a node as an ordered list of pointers (for (B) nodes, we have two such lists) together with a mark which sits on exactly one pointer in each such list. initTree(ϑ) initializes all marks to the first pointer in the respective lists. We will see below how marks are moved to obtain a new solution in each call of getSol$(\cdot)$ from enumerateSolutions.

We now describe function getSol$(\cdot)$ (ignoring the flag $isLast$ and the function hasNextChild$(\cdot)$ which both are described below). Roughly speaking, getSol$(\cdot)$ traverses Υ following the marked pointers using getChildren(n, ϑ) which yields the result of the pointer marked in the list for the current node (for (B) nodes, this method accordingly returns two nodes, one for each of the two child nodes in the tree decomposition). This way, we go down the tree Υ by recursive calls of getSol$(\cdot)$ until the leaves are reached. In the leaves, we start with an empty solution and update it on the way back to the root. The only modifications are done when we are at an (EI) node or at a (B) node.

```
**Program** enumerateSolutions
**begin**
  take node $n$, s.t. $root(n)$
  **for each** $\vartheta$ with $aux_{root}(\vartheta, \_)$ **do**
    initTree($\vartheta$)
    **repeat**
      $(\boldsymbol{X}, isLast) = \mathsf{getSol}(n, \vartheta, true)$
      output $\boldsymbol{X}$
    **until** $isLast$
  **done**
**end**

**Function** hasNextChild$(n, \vartheta)$
*input*: node $n$, type $\vartheta$
*return*: boolean $isLast$
**begin**
  $l$ = number of children of $(n, \vartheta)$
  **if** $(n, \vartheta).mark < l - 1$ **then**
    $(n, \vartheta).mark{+}{+}$
    **return** $true$
  **endif**
  $(n, \vartheta).mark = 0$
  **return** $false$
**end**

**Function** getSol$(n, \vartheta, isLast)$
*input*: node $n$, type $\vartheta$, boolean $isLast$
*return*: tuple of sets $\boldsymbol{X}$, boolean $isLast$
**begin**
  **if** $n$ is (L) node **then**
    $\boldsymbol{X} = (\emptyset, \ldots, \emptyset)$
  **elseif** $n$ is (B) node **then**
    $((n', \vartheta'), (n'', \vartheta'')) = \mathsf{getChildren}(n, \vartheta)$
    $(\boldsymbol{X}', isLast) = \mathsf{getSol}(n', \vartheta', isLast)$
    $(\boldsymbol{X}'', isLast) = \mathsf{getSol}(n'', \vartheta'', isLast)$
    $\boldsymbol{X} = \boldsymbol{X}' \cup \boldsymbol{X}''$
  **elseif** $n$ is (EI) node **then**
    $(n', \vartheta') = \mathsf{getChildren}(n, \vartheta)$
    $(\boldsymbol{X}', isLast) = \mathsf{getSol}(n', \vartheta', isLast)$
    $\boldsymbol{X} = \mathsf{addElement}(n, \vartheta, \boldsymbol{X}')$
  **else** /* (ER) or (P) node */
    $(n', \vartheta') = \mathsf{getChildren}(n, \vartheta)$
    $(\boldsymbol{X}, isLast) = \mathsf{getSol}(n', \vartheta', isLast)$
  **endif**
  **if** $isLast$ **then**
    $isLast = \mathsf{hasNextChild}(n, \vartheta)$
  **endif**
  **return** $(\boldsymbol{X}, isLast)$
**end**
```

Fig. 1. Program enumerateSolutions

For (EI) nodes, addElement$(n, \vartheta, \boldsymbol{X}')$ alters $\boldsymbol{X}'$ according to ε in $aux_I(n, \vartheta, \vartheta', \varepsilon, j)$. For (B) nodes, $\boldsymbol{X}$ is simply the componentwise union of the respective results $\boldsymbol{X}', \boldsymbol{X}''$.

Finally, we describe how the flag $isLast$ works and how the marks on pointers are moved when traversing the tree (this is done in function hasNextChild(n, ϑ)). Flag $isLast$ plays a kind of dual role. Being a parameter, it indicates whether we are currently allowed to move the mark, and this flag is passed down the recursive calls (note that the call from enumerateSolutions always sets this flag to 1, but this is not necessarily the case when going down the second subtree in a (B) node). Being a return value, flag $isLast$ indicates whether all possible positions of marks have been run through in the processed subtree. If this is the case, we may also move the mark of the current node; otherwise we do not touch it. Function hasNextChild(n, ϑ) takes care of the required manipulation of marks: it moves the mark to the next pointer, or in case the mark was already on the last pointer, it moves the mark back to the first pointer of the list; in both cases, the function returns the pointer which is now marked; however, in the former case, it returns as second value $false$, in the latter case (the mark is reset to the first pointer), it returns $true$. Thus, if $true$ is returned to enumerateSolutions we are done, since all possible positions of marks have been run through. Therefore, this algorithm guarantees that (i) each solution leading to the current ϑ in enumerateSolutions is found; (ii) no such solution is returned twice to enumerateSolutions.

Theorem 3. *Let σ and $w \geq 1$ be arbitrary but fixed. For the class C of σ-structures of treewidth w, the solutions of problem* #ENUM-MC(C,MSO) *for input $\mathcal{A} \in C$ and $\varphi \in$ MSO can be enumerated with delay $\mathcal{O}(f(\|\varphi\|, w) \cdot \|\mathcal{A}\|)$, where f is a function that only depends on w and $|\varphi|$.*

Proof. We call the bound $\mathcal{O}(f(\|\varphi\|, w) \cdot \|\mathcal{A}\|)$ fixed-parameter linear (FPL). Since the evaluation of the program Π over $\mathcal{A}$ in the proof of Theorem 1 is FPL, the number of atoms in the LFP of Π over $\mathcal{A}$ is also FPL. Therefore the creation and the size of Υ is FPL. Moreover, a traversal of Υ can also be done in FPL, and enumerateSolutions outputs the solutions with a delay in FPL.

```
Program $\Pi_{3-C}$ (#3-COLORABILITY)
/* (L) node */
$\mathit{solve}(n, \emptyset, \emptyset, \emptyset, 1) \leftarrow \mathit{leaf}(s).$
/* (EI) node */
$\mathit{forbidden}(n, Y) \leftarrow \mathit{bag}(n, X), Y \subseteq X, \mathit{edge}(u, v), u \in Y, v \in Y.$
$\mathit{allowed}(n, Y) \leftarrow$ not $\mathit{forbidden}(n, Y).$
$\mathit{aux}_I(n, R \uplus \{v\}, G, B, R, G, B, j) \leftarrow \mathit{bag}(n, X \uplus \{v\}), \mathit{child}_1(n_1, n), \mathit{bag}(n_1, X),$
    $\mathit{solve}(n_1, R, G, B, j), \mathit{allowed}(n, R \uplus \{v\}).$
$\mathit{aux}_I(n, R, G \uplus \{v\}, B, R, G, B, j) \leftarrow \mathit{bag}(n, X \uplus \{v\}), \mathit{child}_1(n_1, n), \mathit{bag}(n_1, X),$
    $\mathit{solve}(n_1, R, G, B, j), \mathit{allowed}(n, G \uplus \{v\}).$
$\mathit{aux}_I(n, R, G, B \uplus \{v\}, R, G, B, j) \leftarrow \mathit{bag}(n, X \uplus \{v\}), \mathit{child}_1(n_1, n), \mathit{bag}(n_1, X),$
    $\mathit{solve}(n_1, R, G, B, j), \mathit{allowed}(n, B \uplus \{v\}).$
$\mathit{solve}(n, R, G, B, j) \leftarrow \mathit{aux}_I(n, R, G, B, \_, \_, \_, j).$
/* (ER) node */
$\mathit{aux}_R(n, R, G, B, R \uplus \{v\}, G, B, j) \leftarrow \mathit{bag}(n, X), \mathit{child}_1(n_1, n), \mathit{bag}(n_1, X \uplus \{v\}),$
    $\mathit{solve}(n_1, R \uplus \{v\}, G, B, j).$
$\mathit{aux}_R(n, R, G, B, R, G \uplus \{v\}, B, j) \leftarrow \mathit{bag}(n, X), \mathit{child}_1(n_1, n), \mathit{bag}(n_1, X \uplus \{v\}),$
    $\mathit{solve}(n_1, R, G \uplus \{v\}, B, j).$
$\mathit{aux}_R(n, R, G, B, R, G, B \uplus \{v\}, j) \leftarrow \mathit{bag}(n, X), \mathit{child}_1(n_1, n), \mathit{bag}(n_1, X \uplus \{v\}),$
    $\mathit{solve}(n_1, R, G, B \uplus \{v\}, j).$
$\mathit{solve}(n, R, G, B, \mathrm{SUM}(j)) \leftarrow \mathit{aux}_R(n, R, G, B, \_, \_, \_, j).$
/* (B) node */
$\mathit{aux}_B(n, R, G, B, R, G, B, j_1 * j_2) \leftarrow \mathit{bag}(n, X), \mathit{child}_1(n_1, n), \mathit{child}_2(n_2, n), \mathit{bag}(n_1, X),$
    $\mathit{bag}(n_2, X), \mathit{solve}(n_1, R, G, B, j_1), \mathit{solve}(n_2, R, G, B, j_2).$
$\mathit{solve}(n, R, G, B, j) \leftarrow \mathit{aux}_B(n, R, G, B, \_, \_, \_, j).$
/* result (at the root node) */
$\mathit{count}(\mathrm{SUM}(j)) \leftarrow \mathit{root}(n), \mathit{solve}(n, \_, \_, \_, j).$
```

Fig. 2. Counting 3-colorings

5 3-Colorability

In this section, we illustrate our approach with the aid of the counting and enumeration variant of 3-COLORABILITY, whose MSO-encoding $\varphi(R, G, B)$ was given in

Section 3. A graph (V, E) with vertices V and edges E is represented as a σ-structure $\mathcal{A}$ with $\sigma = \{\mathit{edge}\}$. By $\mathcal{A}_{td}$ we denote the σ_{td}-structure which, in addition to the graph, also represents a tree decomposition $\mathcal{T}$ of $\mathcal{A}$ of width $\leq w$ for some constant w. We write $\mathcal{C}(\mathcal{A})$ to denote the set of valid 3-colorings of $\mathcal{A}$, i.e., $\mathcal{C}(\mathcal{A}) = \{(\bar{R}, \bar{G}, \bar{B}) \mid \mathcal{A} \models \varphi(\bar{R}, \bar{G}, \bar{B})\}$. The program in Figure 2 takes a σ_{td}-structure $\mathcal{A}_{td}$ as input and calculates $|\mathcal{C}(\mathcal{A})|$.

Note that the set variables used in Figure 2 are not sets in the general sense, since their cardinality is restricted by the size $w+1$ of the bags, where w is a fixed constant. Hence, these "fixed-size" sets can be simply implemented by means of k-tuples over $\{0, 1\}$ with $k \leq (w+1)$. For the sake of readability, we also use the non-datalog operator $\uplus$ (disjoint union), which could be easily replaced by a "proper" datalog expression. By considering the bags as sets, we no longer need the node type (P).

At the heart of this program is the intensional predicate $\mathit{solve}(n, R, G, B, j)$ with the following intended meaning: n denotes a node in $\mathcal{T}$, the sets R, G, B are the projections of some coloring on $\mathcal{A}_n$ onto $\mathrm{bag}(n)$ and j denotes the number of different colorings on $\mathcal{A}_n$ having this projection. More precisely, let $\Lambda(\mathcal{A}_n, R, G, B) = \{(\bar{R}, \bar{G}, \bar{B}) \in \mathcal{C}(\mathcal{A}_n)$ with projection (R, G, B) onto $\mathrm{bag}(n)\}$, then a fact $\mathit{solve}(n, R, G, B, j)$ shall be in the LFP of $\Pi_{3-C} \cup \mathcal{A}_{td}$, iff $|\Lambda(\mathcal{A}_n, R, G, B)| = j \geq 1$.

The main task of the program is the computation of all facts $\mathit{solve}(n, R, G, B, j)$ via a bottom-up traversal of the tree decomposition. The predicate *count* holds the final result. Π_{3-C} solves the #3-COLORABILITY problem in the following way:

Theorem 4. *Given an instance of* #3-COLORABILITY, *i.e., an* $\{\mathit{edge}\}$*-structure* $\mathcal{A}$*, together with a tree decomposition* $\mathcal{T}$ *of* $\mathcal{A}$ *having width* w*, then* $\mathit{count}(j)$ *with* $j \geq 1$ *is in the LFP of* $\Pi_{3-C} \cup \mathcal{A}_{td}$ *iff* $\mathcal{A}$ *has exactly* j *possible 3-colorings. Moreover, both the construction of* $\mathcal{A}_{td}$ *and the evaluation of* $\Pi_{3-C} \cup \mathcal{A}_{td}$ *can be computed in* $\mathcal{O}(3^{2w+2}) \cdot \|\mathcal{A}\|$*, assuming unit cost for arithmetic operations.*

Proof Idea. The *correctness* of the program Π_{3-C} follows immediately as soon as it has been shown that $\mathit{solve}(n, R, G, B, j)$ indeed has the intended meaning described above. This in turn can be easily shown by structural induction over the tree decomposition $\mathcal{T}$ via a case distinction for the possible node types (L), (EI), (ER), and (B) of n.

For the *complexity*, we notice that program Π_{3-C} is essentially a succinct representation of a quasi-guarded extended datalog program. For instance, in the atom $\mathit{solve}(n, R, G, B, j)$, the sets R, G, and B are subsets of size $\leq w$ of bag A_n at node n. Hence, each combination R, G, B could be represented by three sets $r, s, t \subseteq \{0, \ldots, w\}$ referring to indices of elements in A_n. Hence, $\mathit{solve}(n, R, G, B, j)$ is a succinct representation of constantly many predicates of the form $\mathit{solve}_{r,s,t}(n, j)$. Then $\mathit{bag}(n, X)$ is a quasi-guard in each rule. The *fixed-parameter linearity* follows from Theorem 2. A finer analysis of the program reveals, that for #3-COLORABILITY the function $f(\|\varphi\|, w)$ can be explicitly stated as 3^{2w+2}, since there are at most 3^{w+1} partitions R, G, B at each node n. Hence, the combination of two partitions as the argument of predicates (e.g., $\mathit{aux}_I(n, R, G, B, R', G', B', j)$) yield at most 3^{2w+2} groundings of these predicates. □

The enumeration variant of 3-COLORABILITY can be solved with linear delay by a slight modification of the program in Figure 1. Instead of nodes (n, ϑ) in Υ, we consider nodes of the form (n, R, G, B). Then we again use the *child*$(\cdot)$ and *aux*$_\lambda(\cdot)$ facts in the LFP of $\Pi_{3-C} \cup \mathcal{A}_{td}$ to establish a parent-child relation between nodes (n, R, G, B) and (n', R', G', B'). Analogously to the program in Figure 1, each solution $\bar{R}, \bar{G}, \bar{B}$ is constructed in a bottom-up way, starting with an empty solution at the leaves. When we reach an (EI) node n, the set difference between R, G, B and R', G', B' of a fact *aux*$_I(n, R, G, B, R', G', B', j)$ determines how the sets $\bar{R}, \bar{G}, \bar{B}$ are extended.

Discussion. The generic construction of a program Π from some MSO-formula φ in the proof of Theorem 1 is "constructive" in theory but not feasible in practice. However, in contrast to the MSO-to-FTA approach, datalog allows us to construct tailor-made programs like program Π_{3-C}, which follow the intuition of the generic programs Π but incorporates several short-cuts that make it indeed feasible: Above all, as the intended meaning of the *solve*-predicate suggests, we only propagate those "types" (represented by the *solve*-facts) which can possibly be extended in bottom-up direction to a solution. Moreover, the *solve*-facts do not exactly correspond to the types in Theorem 1 but only describe the properties of each type which are crucial for the target formula $\varphi(R, G, B)$.

6 Conclusion

We have extended the monadic datalog approach [14] to MSO-definable counting and enumeration problems. For the latter, we have thus shown that they can be solved with linear delay in case of bounded treewidth. We have also illustrated the potential of our approach for constructing efficient algorithms by solving the counting and enumeration variant of 3-COLORABILITY. As future work, we plan to apply our approach to further MSO-definable problems. Finally, we also want to tackle extensions of MSO with our approach; in particular extensions by optimization (i.e., counting/enumerating the *minimal* or the *maximal* solutions only) [2] and by local cardinality constraints [22].

References

1. Foto N. Afrati and Rada Chirkova. Selecting and using views to compute aggregate queries. In *Proc. ICDT'05*, volume 3363 of *LNCS*, pages 383–397. Springer, 2005.
2. Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12(2):308–340, 1991.
3. Guillaume Bagan. MSO queries on tree decomposable structures are computable with linear delay. In *Proc. CSL'06*, volume 4207 of *LNCS*, pages 167–181. Springer, 2006.
4. Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.
5. Stefano Ceri, Georg Gottlob, and Letizia Tanca. *Logic Programming and Databases*. Springer, 1990.
6. Sara Cohen, Werner Nutt, and Alexander Serebrenik. Rewriting aggregate queries using views. In *Proc. PODS'99*, pages 155–166. ACM, 1999.
7. Bruno Courcelle. Graph rewriting: An algebraic and logic approach. In *Handbook of Theoretical Computer Science, Volume B*, pages 193–242. Elsevier Science Publishers, 1990.
8. Bruno Courcelle. Linear delay enumeration and monadic second-order logic. *Discrete Applied Mathematics*, 157(12):2675–2700, 2009.

9. Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Springer, New York, 1999.
10. T. Eiter, W. Faber, M. Fink, and S. Woltran. Complexity results for answer set programming with bounded predicate arities and implications. *Annals of Mathematics and Artificial Intelligence*, 51(2–4):123–165, 2007.
11. Jörg Flum, Markus Frick, and Martin Grohe. Query evaluation via tree-decompositions. *Journal of the ACM*, 49(6):716–752, 2002.
12. Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Springer, 2006.
13. M. Frick and M. Grohe. The complexity of first-order and monadic second-order logic revisited. In *Proc. LICS'02*, pages 215–224, 2002.
14. Georg Gottlob, Reinhard Pichler, and Fang Wei. Monadic datalog over finite structures with bounded treewidth. In *Proc. PODS'07*, pages 165–174. ACM, 2007.
15. Martin Grohe. Descriptive and parameterized complexity. In *Proc. CSL'99*, volume 1683 of *LNCS*, pages 14–31, 1999.
16. Stéphane Grumbach, Maurizio Rafanelli, and Leonardo Tininini. On the equivalence and rewriting of aggregate queries. *Acta Inf.*, 40(8):529–584, 2004.
17. Michael Jakl, Reinhard Pichler, Stefan Rümmele, and Stefan Woltran. Fast counting with bounded treewidth. In *Proc. LPAR'08*, volume 5330 of *LNCS*, pages 436–450. Springer, 2008.
18. David B. Kemp and Peter J. Stuckey. Semantics of logic programs with aggregates. In *Proc. ISLP*, pages 387–401, 1991.
19. Ton Kloks. *Treewidth, Computations and Approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer, 1994.
20. Anthony C. Klug. Equivalence of relational algebra and relational calculus query languages having aggregate functions. *J. ACM*, 29(3):699–717, 1982.
21. Leonid Libkin. *Elements of Finite Model Theory*. Springer, 2004.
22. Stefan Szeider. Monadic second order logic on graphs with local cardinality constraints. In *Proc. MFCS'08*, volume 5162 of *LNCS*, pages 601–612. Springer, 2008.
23. Moshe Y. Vardi. The complexity of relational query languages (extended abstract). In *Proc. STOC'82*, pages 137–146. ACM, 1982.