Automata-theoretic and Datalog-based solutions of Monadic Second-order Logic Evaluation Problems over Structures of bounded-treewidth *

Eugénie Foustoucos† Labrini Kalantzi‡

February 2011

Abstract

We propose automata-theoretic and datalog-based solutions for the Monadic Second Order (MSO) evaluation problem over finite structures of bounded treewidth, and then extend this approach to MSO-definable optimization problems. More precisely, we introduce decomposition-automata which can be thought as a generalization of assignment automata defined in [14]; these automata, running over tree-decompositions of an input structure, directly compute solutions to the considered MSO evaluation problems. The constructive proof of this result provides a direct reduction of the initial MSO evaluation problem to a decomposition-automata evaluation problem. We then use datalog and its optimization techniques to implement the computation mechanism of decomposition automata in order to provide optimized datalog solutions for the initial MSO evaluation problems. Since the automata construction can be completely expressed in datalog, we show that given an MSO formula we can directly define datalog queries that compute the solutions to the considered problems. The resulting datalog programs prove that k-ary MSO-definable queries over structures of bounded-treewidth are definable in datalog of arity k + 1, generalizing the result of [17] that unary MSO-definable queries are monadic datalog definable, and extending the corresponding result of [14] proven for the case of trees. Finally, we illustrate our approach by applying it in order to solve VERTEX COVER and related optimization problems.

1 Introduction

Treewidth is a concept of great importance in computer science, since many NP-hard algorithmic problems become fixed-parameter tractable when parameterized by the tree-width of the input structure (see [11, 13]). A very powerful and general such result is Courcelle’s theorem [6] stating that any property of graphs definable in MSO can be decided in linear time on any class of graphs of bounded treewidth, or in other words, MSO is fixed-parameter tractable in linear time on any such class of graphs. The standard proof consists in the construction of a finite bottom-up

---

*A preliminary form of the results of this work was presented at the "2nd Workshop on Graph Decompositions Theoretical, Algorithmic and Logical Aspects", October 18 - 22, 2010 CIRM, Marseille, France [21].

†MPLA, Department of Mathematics, National and Capodistrian University of Athens and Department of Computer Science, Athens University of Economics and Business, aflow@otenet.gr, eugenie@aueb.gr.

‡MPLA, Department of Mathematics, National and Capodistrian University of Athens. lkalan@math.uoa.gr. The research of this author has been partially supported by the University of Patras, Project Karatheodory "Basic Research 2007-10: Logic and Theory of Algorithms".
tree automaton that recognizes a tree-decomposition of the input structure. Automata-theoretic based algorithms are also proposed in [12], solving the MSO evaluation problem on structures of bounded treewidth in time linear in the size of the input structure plus the size of the output. This approach, based on [2] and [9], reduces the initial evaluation problem into an equivalent MSO evaluation problem over colored binary trees; the solution to the latter problem is based on the well-known MSO-automata connection stated in Doner [10], Thatcher & Wright [20] theorem.

In [16], Gottlob and Koch study the fragment of MSO over trees that consists of MSO formulas of the form $\phi(x)$, i.e. with one free first-order variable, and they show that it is equivalent to monadic datalog over trees in its ability to express unary queries over trees (ranked as well as unranked). Their proof proceeds as follows: the first direction, i.e. that each monadic datalog query is MSO-definable via a formula $\phi(x)$, is part of the database folklore. The proof of the other direction, i.e. that each unary MSO-definable query (over trees) can be expressed in monadic datalog, is based on the Feferman-Vaught composition method (see [18] for an overview of this method). In [17], Gottlob et al. prove that every MSO-definable unary query over finite structures of bounded treewidth is also definable in monadic datalog, generalizing thus the corresponding expressivity result of [16] for finite trees. As in the case of trees, their approach is based on versions of Feferman-Vaught style composition theorems. In [19], Pichler et al. extend the datalog approach of [17] to solve the MSO counting problem over structures of bounded treewidth. The latter problems consists in the computation of the number of the satisfying assignments of an MSO formula $\phi$ over a structure. In [14], we extended the datalog solution of [16] for evaluating MSO formulas $\phi(x)$ over trees to the full class of MSO formulas $\phi(x_1, \ldots, x_\ell, X_1, \ldots, X_j)$ and we proved that $k$-ary MSO-definable queries over finite trees are definable in datalog of arity $k + 1$. Unlike the works mentioned above that are based on the transfer of the ideas of the Feferman-Vaught composition method in a datalog framework, our work was based on the transfer of the automata-MSO connection.

In this work, we generalize the automata-theoretic and datalog-based solution of [14], for the MSO evaluation problem over trees, to structures of bounded treewidth. And so, we also extend the datalog solution of [17] concerning the restricted fragment of MSO consisting of formulas of the form $\phi(x)$ to whole MSO. Moreover, apart from dealing with the MSO evaluation problem, we consider three optimization problems defined via MSO formulas $\phi(X)$ on such classes of structures. Evaluation of these three problems, that we call MSO-definable optimization problems, consists in i) answering whether there exists a set $X$ of size $k$ satisfying MSO formula $\phi(X)$, ii) computing the cardinality of satisfying assignments for $\phi(X)$ having minimum (resp. maximum) size and iii) computing the satisfying assignments for $\phi(X)$ of minimum (resp. maximum) size. In all aforementioned problems, besides an MSO formula $\phi$ and a finite structure $A$, we consider as part of our input a tree decomposition $I$ of $A$; we propose automata-theoretic and datalog-based solutions for these problems too.

In the first part of this work (Section 3), we present a direct reduction of initial MSO evaluation problem over structures of bounded treewidth to an equivalent tree-automata evaluation problem. More precisely, we define new tree-automata formalism, called decomposition automata that run over decomposed structures. A decomposed structure, defined with respect to a tree-decomposition that is also part of our initial input, is constructed via a proper extension of the input tree-decomposition with parts encoding the relations of the input structure. That is,
it constitutes a tree-like encoding of the input structure and its given tree-decomposition. The definition of these automata is based on the development of a proper encoding of assignments over tree-decompositions; more precisely, decomposition automata, running over decomposed structures, produce via their successful runs encoded satisfying assignments of the initial structure. We prove that for any MSO formula \( \phi \) there exists a decomposition automaton producing the satisfying assignments of \( \phi \) over any decomposed structure of bounded treewidth. The proof is constructive, in the sense that explicit definitions of decomposition automata are given, and it constitutes a different automata-based proof of Courcelle’s theorem extension given in [12].

An advantage of the proposed decomposition automata approach, for the solution of the MSO evaluation problem over structures of bounded treewidth, is that a transformation of the initial MSO formula over the input structure into a new MSO formula over trees is not required; in such cases (see, e.g., the approach of [12]), the automaton construction for the new formula, which is significantly more complex than the initial formula, would require much more steps and more complex constructions. However, since we consider all MSO formulas, the “state explosion” of the resulting automata is in general inevitable [15]. Thus, the required space might become a problem in practical applications. Courcelle and Durand address this problem in [7, 8] proposing, among others, fragments of MSO having interesting expressive power, and for which the automata constructions are tractable. Although their approaches are presented with respect to graphs of bounded tree-width, their ideas are as well applied in our context providing possible solutions to practical space limitations problems. However, the study of optimizations concerning the automata construction is beyond the scope of this work. Nevertheless, the importance of the general automata-theoretic method should not be underestimated; besides the existence of formula that in practice do not cause state explosion, the general method can provide solutions in cases where direct algorithms do not exist for a specific problem. Moreover, we believe that the existence of a general solution for a problem, can work as a basis setting the main directions for the derivation of an efficient algorithm.

Decomposition-automata (together with their counting version) provide a new automata-theoretic formalism to define MSO-related queries (i.e. the queries involved in the various MSO evaluation problems defined above); however, we still need to have efficient algorithms to implement the evaluation procedure of such automata (i.e to compute the queries that these automata define). In the second part of this work (Section 4), we propose datalog, the query language for deductive databases, as a tool providing optimized algorithms for the evaluation of decomposition-automata. In particular, we systematically generalize the datalog approach of [14] to MSO evaluation and MSO-definable optimization problems over structures of bounded treewidth. More precisely, we express the decomposition-automata evaluation problem into a datalog framework and then we use datalog optimization methods to evaluate the corresponding datalog queries. The resulting datalog algorithms propose efficient evaluation procedures for the decomposition-automata evaluation problem and thus also for the initial considered MSO evaluations problems. Although the resulting datalog solution is based on the proper adaptation and extension of the corresponding solution of [14] we had to overcome several difficulties in order to obtain it. One of the main reason for this is that in the case of structures of bounded treewidth.

\[1\] A more intuitive way to think of these automata is via their connection to automata accepting tree-decompositions encoding satisfying assignments: the latter can be thought as a different representation of the former. There exists an one-to-one correspondence between runs of these two automata.
we have two input databases, corresponding to the structure and the tree-decomposition; thus, we moreover have to i) define in datalog the input tree of our automata and ii) incorporate the more involved decoding definition that decomposition automata have. More precisely, we derive naturally a direct datalog solution, due to i) our definition of the decomposition automata (i.e. the definition of the encoding function involved) and the presentation of the automata-theoretic reduction via them and ii) our representation of the tree-decomposition as a database. Notice that since the automata construction can be completely expressed in datalog, for a given MSO formula we can directly define datalog queries that compute the solutions of the corresponding evaluation problems.

Besides the algorithmic importance of this result due to the operational semantics of datalog, we moreover prove new interesting expressivity results concerning datalog. More precisely, we prove the \((k + 1)\)-datalog definability of \(k\)-ary MSO-definable queries over structures of bounded treewidth generalizing the corresponding result of [14] for the case of trees and extending the result of [17] that unary MSO-queries are monadic datalog definable over structures of bounded treewidth (which we also reprove).

Our approach (i.e. the construction of decomposition-automata together with the corresponding datalog rules that evaluate them) constitutes a complete and general algorithmic procedure providing a solution to all problems formalized as MSO evaluation problems on structures of tree-decomposition, called \(A\)-special tree-decomposition. More precisely, we derive naturally a direct datalog solution, due to i) our definition of the decomposition automata (i.e. the definition of the encoding function involved) and the presentation of the automata-theoretic reduction via them and ii) our representation of the tree-decomposition as a database. Notice that since the automata construction can be completely expressed in datalog, for a given MSO formula we can directly define datalog queries that compute the solutions of the corresponding evaluation problems.

Besides the algorithmic importance of this result due to the operational semantics of datalog, we moreover prove new interesting expressivity results concerning datalog. More precisely, we prove the \((k + 1)\)-datalog definability of \(k\)-ary MSO-definable queries over structures of bounded treewidth generalizing the corresponding result of [14] for the case of trees and extending the result of [17] that unary MSO-queries are monadic datalog definable over structures of bounded treewidth (which we also reprove).

Our approach (i.e. the construction of decomposition-automata together with the corresponding datalog rules that evaluate them) constitutes a complete and general algorithmic procedure providing a solution to all problems formalized as MSO evaluation problems on structures of bounded-treewidth. We choose to illustrate our approach by applying it in order to solve the following problems: **vertex cover, minimum vertex cover, minimum vertex cover cardinality and \(k\)-vertex cover.**

## 2 Preliminaries

**Structures, MSO, tree-decompositions**  A (relational) signature \(\tau\) is a finite set of predicate symbols \(R_1, \ldots, R_\ell\); every predicate symbol \(R_i, i \in \{1, \ldots, \ell\}\), is equipped with a natural number \(r_i \geq 0\) called its arity. A \(\tau\)-structure \(A\) is a pair \((A, (R_i^A)_{i \in \{1, \ldots, \ell\}})\), where \(A\) is a nonempty set called **domain**, and \(R_i^A\) is a relation of arity \(r_i\) on \(A\). A structure \(A\) is **finite** if \(A\) is a finite set.

**Monadic second-order logic** (or in short MSO) is an extension of first-order logic (FO) with set variables on which quantification is allowed. In this work we deal with the class MSO[\(\tau\)] of all MSO-formulas of vocabulary \(\tau\). Atomic formulas of MSO[\(\tau\)] are \(R_i(x_1, \ldots, x_{r_i})\) for all \(R_i \in \tau\) and \(\text{In}(x, Z)\) interpreted as “the element \(x\) belongs to the set \(Z\)”; non atomic formulas are of the form \(\exists X \phi, \neg \phi, \phi \land \psi\) where \(\phi\) and \(\psi\) are MSO[\(\tau\)] formulas.

A **tree-decomposition** of a \(\tau\)-structure \(A\) is a pair \((T, (A_n)_{n \in T})\) where \(T\) is a tree with domain \(T\) and each \(A_n\), called **bag** of the decomposition, is a subset of the domain \(A\) of \(A\) such that

i) for every \(a \in A\), the set \(T_a = \{n \in T \mid a \in A_n\}\) is nonempty and connected in \(T\);

ii) for every \((a_1, \ldots, a_{r_i}) \in R_i^A, 1 \leq i \leq \ell\), there is an \(n \in T\) such that \((a_1, \ldots, a_{r_i}) \subseteq A_n\).

The **width** of a tree decomposition \((T, (A_n)_{n \in T})\) is the number \(\max\{|A_n| \mid n \in T\} - 1\). The treewidth \(\text{tw}(A)\) of structure \(A\) is the minimum of the widths of the tree decompositions of \(A\).

It is more convenient in our framework to have representations of the bags of the tree decompositions is the form of tuples instead of sets. Thus, we consider a slightly different form of tree-decomposition, called **special tree-decomposition** of a structure \(A\) with treewidth \(w\), which is an **ordered** tree-decomposition defined by pair \((T, (\bar{a}_n)_{n \in T})\) where \(T\) is a binary tree, each
\[ \pi_n \text{ is a } u \text{-tuple } (a_1, \ldots, a_u), \ 1 \leq u \leq w + 1, \text{ of elements of } A \text{ and } (T_s, (a_n)_{n \in T}), \ a_n = \{a_1, \ldots, a_u\}, \text{ is a tree decomposition of } A \text{ of width } w. \text{ The transformation of a given tree-decomposition to a special tree-decomposition of the same width can be performed in linear time.} \]

**Example 2.1.** Let \( G \) be an \( E \)-structure corresponding to an undirected graph with vertex set \( V = \{a, b, c, d, e, f, g, h\} \) (\( E \) is a binary predicate interpreted as the symmetric edge relation) and \( E^G = \{(a, b), (a, c), (a, g), (b, c), (c, d), (c, e), (c, f), (d, e), (f, g), (g, h), (b, a), \ldots, (h, g)\} \). The tree-decomposition \( \mathcal{I} \) given in Figure 1 is a special tree-decomposition of width 2 of \( G \) (the elements of the blocks are considered ordered according to the way they are listed).

![Graph G and a special tree-decomposition I for G.](image)

We say that a class \( \mathcal{C} \) of structures is of bounded treewidth if there exists a \( w \geq 1 \) such that for every structure \( A \in \mathcal{C} \) the treewidth \( tw(A) \) of \( A \) is at most \( w \). In this work, we consider classes of finite \( \tau \)-structures of bounded treewidth. Notice that, as stated in the following theorem, tree-decompositions for structures of bounded treewidth can be computed efficiently.

**Theorem 2.1 (\cite{5}).** There is a polynomial \( p(x) \) and an algorithm that, given a structure \( A \), computes a tree-decomposition of \( A \) of width \( w = tw(A) \) in time \( 2^{p(w)} \cdot |A| \).

**Trees, tree-automata, MSO-automata connection over trees** Let \( \Gamma \) be a finite set of symbols called the alphabet of colors and let \( \tau_{\Gamma} \) be the signature \( \tau_{\Gamma} = \{S_1, S_2, (P_\gamma)_{\gamma \in \Gamma}\} \) where \( S_1, S_2 \) are binary predicate symbols and \( (P_\gamma)_{\gamma \in \Gamma} \) are unary predicate symbols. A \( \Gamma \)-colored tree (or \( \Gamma \)-tree) \( T \) is a finite relational \( \tau_{\Gamma} \)-structure \( (T, S_1^T, S_2^T, (P_\gamma^T)_{\gamma \in \Gamma}) \) such that \( (n, n_i) \in S_1^T \) iff \( n_i \) is the \( i \)-th child of \( n \), \( i = 1, 2 \), and \( n \in P_\gamma^T \) iff \( \gamma \) is the color of \( n \). The elements of \( T \) are called nodes of tree \( T \). We can equivalently represent \( T \) as a pair \( (t, c) \), where \( t = (T, s_1, s_2) \) is the underlying tree of \( T \) \((s_1, s_2)\) are the left and right child functions respectively) and where \( c : T \to \Gamma \) is a coloring function for \( t \) satisfying \( c(n) = \gamma \) iff \( n \in P_\gamma^T \). A subtree \( T' \) of \( T \) is a connected subgraph of \( T \) (with set of nodes \( T' \subseteq T \)): \( T' = (T', S_1^T \cap T'^2, S_2^T \cap T'^2, (P_\gamma^T \cap T')_{\gamma \in \Gamma}) \). We denote \( T_n \) the subtree of \( T \) rooted at \( n \) with domain \( T_n \) containing all the leaves reachable from \( n \); such subtrees are called full.

A non deterministic bottom-up \( \Gamma \)-colored tree automaton (\( \Gamma \)-automaton) \( A \) is a tuple \((\Gamma, Q, \Delta_0, \Delta, F)\), where \( \Gamma \) is a finite alphabet of colors, \( Q \) is a finite set of states, \( F \subseteq Q \) is the set of final states and \( \Delta_0, \Delta \) are relations: \( \Delta_0 \subseteq \Gamma \times Q \) and \( \Delta \subseteq Q \times Q \times \Gamma \times Q \). Automaton \( A \) is deterministic if relations \( \Delta_0, \Delta \) define mappings \( \delta_0 : \Gamma \to Q, \delta : Q \times Q \times \Gamma \to Q \). A run \( \rho \) of an (either deterministic or not) automaton \( A \) on a \( \Gamma \)-tree \( T \), is a mapping, assigning states to nodes s.t. i) if
n is a leaf with color a, then ρ(n) = q if there exists a transition (a, q) ∈ ∆₀, and ii) if n is a node of color a, having children n₁, n₂, then the value ρ(n) is such that (ρ(n₁), ρ(n₂), a, ρ(n)) ∈ ∆. A run is successful if it maps the root to a final state. A Γ-tree T is recognized by a Γ-automaton A if there exists a successful run of A on T. A class C of Γ-trees is recognizable if there exists an automaton that recognizes exactly those Γ-trees that belong to C; class C is MSO[Γ]-definable if there exists an MSO[Γ] sentence φ such that for every Γ-tree T, φ is true on T iff T is in C.

**Theorem 2.2 (Doner [10], Thatcher & Wright [20]).** A class of Γ-trees T ⊆ TΓ is recognizable if and only if is MSO[Γ]-definable.

**MSO-definable queries - The MSO evaluation problem & MSO-definable optimization problems** Each MSO[τ] formula φ(y₁, ..., yₖ, x₁, ..., xₖ) defines a mapping, also denoted by φ, over a class Cτ of finite τ-structures: for each τ-structure A with domain A,

φ(A) = {(a₁, ..., aₖ, B₁, ..., Bₖ) | A |= φ(a₁, ..., aₖ, B₁, ..., Bₖ)} ⊆ A¹ × (℘(A))ₖ;

the class of mappings q over Cτ defined by some MSO formula with k free variables (i.e. such that there exists an MSO formula φ with q(A) = φ(A) for all τ-structures A) is called the class of k-ary MSO-definable queries over the class Cτ. The MSO evaluation problem on Cτ is the problem of computing φ(A) given MSO[τ] formula φ and τ-structure A; obviously, by solving this problem we can also solve the problem of evaluating MSO queries over Cτ.

Besides the MSO evaluation problem, we separately consider MSO-definable optimization problems on finite structures of bounded treewidth, i.e. the following three evaluation problems defined with respect to MSO formulas φ(X):

i) the problem of deciding, for an MSO-formula φ(X) and a structure A whether φ has a satisfying assignment of size k

ii) the problem of computing, for an MSO-formula φ(X) and a structure A, the cardinality of those elements of φ(A) having minimum (resp. maximum) size and

iii) the problem of computing, for an MSO-formula φ(X) and a structure A, the elements of φ(A) of minimum (resp. maximum) size.

**Datalog preliminaries** In the datalog context, a database of domain D is a finite relational structure D = (D, {r₁, ..., rₙ}) of signature {R₁, ..., Rₙ} (i.e. D is a finite set and each Rᵢ is a predicate symbol of arity aᵢ, naming relation rᵢ over D): we shall equivalently represent D as a set of Rᵢ-facts (i.e. expressions of the form Rᵢ(c₁, ..., cₙ), cⱼ ∈ D), i = 1, ..., n, i.e. D ⊆ ∪ᵢ{Rᵢ(σ) | σ ∈ rᵢ ⊆ Dⁿ}. A datalog program is a collection of rules of the form A₀ ← A₁, ..., Aₖ where A₀ is the head, and A₁, ..., Aₖ, form the body of the rule. Each Aᵢ is an atom i.e. an expression of the form R(x₁, ..., xₙ) where the xᵢ’s are either variables or constants and R is a predicate symbol (we say predicate for short); each variable occurring in the head must have at least one occurrence in the body. Predicates appearing in rule heads name relations defined by the program and are called intensional database (IDB) predicates while predicates not appearing in rule heads are called extensional database (EDB) predicates; the set IDB(Π) (resp. EDB(Π)) of IDB (resp. EDB) predicates of program Π constitutes the intensional (resp. extensional) schema of Π. The set of constants appearing in Π is denoted by Const(Π). We call input database for program Π any database with signature EDB(Π).
Given a domain $D$ and a given finite set $V$ of variables, we call $D$-valuation $v$ over $V$ a total mapping from $V$ to $D$; we call $D$-instantiation (or simply instantiation) of a rule $r$ the result of applying to $r$ a $D$-valuation $v$ over the set of variables occurring in $r$, i.e. the result of replacing each variable $x$ in $r$ by $v(x)$. In the following, $\Pi$ denotes a program, $\mathcal{D}$ denotes an input database of domain $D$ for $\Pi$ and we refer to them by the pair $(\Pi, \mathcal{D})$. We say that the fact $P(\pi), \pi = (a_1, \ldots, a_m), a_i \in D \cup \text{Const}(\Pi)$, is derivable from $(\Pi, \mathcal{D})$ (or is computed by $\Pi$ on input $\mathcal{D}$), and we write $(\Pi, \mathcal{D}) \models P(\pi)$, if either $P(\pi) \in \mathcal{D}$ or there exists a $D$-instantiation $r'$ of a rule $r$ of $\Pi$ such that $P(\pi)$ is the head of $r'$ and every fact in the body of $r'$ is derivable from $(\Pi, \mathcal{D})$.

Given $(\Pi, \mathcal{D})$, we define the semantics of $\Pi$ on $\mathcal{D}$, denoted $\Pi(\mathcal{D})$, as the set of facts derivable from $(\Pi, \mathcal{D})$; $\Pi(\mathcal{D})$ is finite and it constitutes a database over domain $D \cup \text{Const}(\Pi)$ and of signature $\text{EDB}(\Pi) \cup \text{IDB}(\Pi)$, s.t. $\Pi(D)$, when restricted to signature $\text{EDB}(\Pi)$, coincides with $D^2$. Clearly, each program $\Pi$ defines a mapping from input databases of signature $\text{EDB}(\Pi)$ to output databases of signature $\text{EDB}(\Pi) \cup \text{IDB}(\Pi)$ such that the image of $\mathcal{D}$ is $\Pi(\mathcal{D})$. A datalog query $Q$ is a pair $(\Pi, Q)$ consisting of a datalog program $\Pi$ together with an IDB predicate $Q \in \text{IDB}(\Pi)$ called goal predicate; $Q$ defines a mapping from input databases for $\Pi$ to relations named by the predicate $Q$, such that, for every input database $\mathcal{D}$, $Q(\mathcal{D})$ is the set of $Q$-facts derivable from $(\Pi, \mathcal{D})$ (clearly $Q(\mathcal{D}) \subseteq \Pi(\mathcal{D})$); we also consider $Q(\mathcal{D}) = \{ \pi \in D \cup \text{Const}(\Pi) \mid (\Pi, \mathcal{D}) \models Q(\pi)\}$ (especially in the statements of our propositions) and we say that $Q$ on input $\mathcal{D}$ computes $Q(\mathcal{D})$. Two datalog queries $(\Pi, Q)$ and $(\Pi', Q')$ are equivalent if they define the same mapping. We call arity of program $\Pi$ the maximum arity of the IDB predicates of $\Pi$; the arity of query $(\Pi, Q)$ is the arity of $\Pi$. A datalog program/query is monadic if its arity is 1.

The above given recursive definition of derivable facts from $(\Pi, \mathcal{D})$ yields the following algorithm (known as bottom-up evaluation) for computing $\Pi(\mathcal{D})$ in a finite number of steps starting from input database $\mathcal{D}$: first we set $\mathcal{D}^0 = \mathcal{D}$; then, at each step $i + 1 > 0$ we produce a new database $\mathcal{D}^{i+1}$, by enriching current database $\mathcal{D}^i$ with the heads of those rule instantiations of $\Pi$ having in their body only facts contained in $\mathcal{D}^i$. After $k$ steps, for some $k$, we will finally have $\mathcal{D}^k = \mathcal{D}^{k+1}$ and $\Pi(\mathcal{D}) = \mathcal{D}^k$; clearly $k \leq |\mathcal{D}|^{\text{ar}(\Pi)}$ where $\text{ar}(\Pi)$ is the arity of $\Pi$. We call active instantiations of $(\Pi, \mathcal{D})$ the instantiations of $\Pi$ participating in the computation of $\Pi(\mathcal{D})$ during bottom-up evaluation; and we measure the complexity of evaluating program $\Pi$ on input $\mathcal{D}$ by the number of active instantiations of $(\Pi, \mathcal{D})$.

Let $Q = (\Pi, Q)$ be a query such that, for every database $\mathcal{D}$, $Q(\mathcal{D}) \in \mathcal{D}^k \setminus \mathcal{D}^{k-1}$ where $\mathcal{D}^k = \Pi(\mathcal{D})$ as above. For such queries (as the ones defined in this work) we compute $Q(\mathcal{D})$ via bottom-up evaluation; thus we measure the complexity of evaluating $Q(\mathcal{D})$ by the number of active instantiations of $(\Pi, \mathcal{D})$. The active instantiations of $(\Pi, \mathcal{D})$ can be partitioned into the set of $Q$-relevant instantiations w.r.t. $\mathcal{D}$ (an instantiation is called $Q$-relevant if it participates in the derivation of some fact in $Q(\mathcal{D})$) and the set of useless instantiations (w.r.t. $Q$ and $\mathcal{D}$). The optimization of bottom-up evaluation consists in eliminating the useless instantiations (w.r.t. $Q$ and $\mathcal{D}$) as much as possible. This can be achieved by rewriting $Q$ into an equivalent query $Q'$ using a rewriting optimization algorithm (such as resolution-filtering method, referred in the literature as “magic sets” method [3]) which proceeds in two steps: first it defines new predicates,

---

²The rules of $\Pi$ are interpreted as logical formulas universally quantified, thus $\Pi(\mathcal{D})$ is a model of $\Pi$ containing $\mathcal{D}$; $P(\pi)$ is true in model $\Pi(\mathcal{D})$ if $P(\pi)$ is derivable from $(\Pi, \mathcal{D})$ (i.e. $\Pi(\mathcal{D}) \models P(\pi)$ if $(\Pi, \mathcal{D}) \models P(\pi)$).
called filter predicates, using a filter program \( F \) for \( Q = (\Pi, Q) \); second it rewrites the initial program \( \Pi \) into a new program \( \Pi^F \) by adding, in an adequate way, the filter predicates in the body of rules of \( \Pi \). Program \( \Pi^F \) filters the set of active instantiations of the initial program \( \Pi \) by eliminating instantiations that are not \( Q \)-relevant while conserving all \( Q \)-relevant instantiations, thus queries \( Q \) and \( Q' = (F \cup \Pi^F, Q) \) are equivalent (we say that \( \Pi \) is optimized into \( F \cup \Pi^F \)). We say that filter \( F \) is optimum for \( Q \) when \( F \) manages to eliminate all useless instantiations i.e. when, for all input databases \( D \), the active instantiations of \((\Pi^F, D \cup F(D))\) correspond exactly to the \( Q \)-relevant instantiations of \( \Pi \) w.r.t. \( D \) (that is, by deleting all occurrences of filter facts from the former set of instantiations, we obtain the latter). When filter \( F \) is optimum we say that \( Q \) is optimally rewritten into \( Q' \).

A mapping (resp. boolean mapping) \( f \) over a class \( C_\tau \) of finite \( \tau \)-structures is called a-datalog definable if there exists a datalog query \( Q_f \) of arity \( a \) such that, for every \( \tau \)-structure \( A \in C_\tau \), \( f(A) = Q_f(D_A) \) (resp. \( f(A) = 1 \) iff \( Q_f(D_A) \neq \emptyset \)), where database \( D_A \) corresponds to structure \( A \). Recall that in this work we consider classes of \( \tau \)-structures having bounded treewidth. So, besides the input structure \( A \) we shall always consider as part of the input a special tree-decomposition \( \mathcal{I} \) of \( A \); and thus \( D_A \) in the above definability definition should be always considered containing facts encoding the input tree-decomposition \( \mathcal{I} \) of \( A \).

By allowing negated atoms in the body of datalog rules, we obtain datalog with negation programs, called datalog\textsuperscript{\neg} programs (see [1, 4] for details). A datalog\textsuperscript{\neg} program \( \Pi \) is called stratified if there exists a partition \( \{\Pi_1, \ldots, \Pi_n\} \) of \( \Pi \) such that each IDB predicate is defined by a unique subprogram \( \Pi_i \) and such that \( \Pi_i \)'s (called the strata of \( \Pi \)) can be ordered is such a way that if IDB predicate \( P' \) (resp. the negation of \( P' \)), defined by \( \Pi_i \), is used in the definition of IDB predicate \( P \), defined by \( \Pi_k \), then \( \Pi_j \leq \Pi_k \) (resp. \( \Pi_j < \Pi_k \)); let \( \Pi_1 < \cdots < \Pi_n \) be such an ordering, the semantics of stratified datalog\textsuperscript{\neg} program \( \Pi \) is defined as follows: \( \Pi(D) = D \cup \bigcup_{1 \leq j \leq n} \Pi'_j(D) \) where, for all \( i \), \( P(\overline{a}) \in \Pi'_i(D) \) if there exists an instantiation \( r' \) of a rule \( r \) of \( \Pi_i \) such that \( P(\overline{a}) \) is the head of \( r' \) and each fact \( P'(\overline{b}) \) (resp. negated fact \( \neg P'(\overline{b}) \)) in the body of \( r' \) is contained in \( D \cup \bigcup_{j \leq i} \Pi'_j(D) \) (resp. is such that \( P'(\overline{b}) \) is not contained in \( D \cup \bigcup_{j < i} \Pi'_j(D) \)). Notice that a datalog\textsuperscript{\neg} program with negation on EDB predicates only, is a special case of stratified datalog\textsuperscript{\neg} program having as single stratum itself. A datalog\textsuperscript{\neg} program with inequality and negation on EDB predicates only is called semipositive datalog program.

3 The MSO-automata connection for structures of bounded treewidth

In this section we present a direct reduction of the MSO evaluation problem over structures of bounded treewidth to an equivalent tree-automata evaluation problem. The main points of this reduction are i) the definition of a tree-representation of the input structure and ii) the definition of a proper encoding of assignments over the considered tree-representation of the structure. The tree-representation of the input structure is based on the input tree-decomposition, which is extended to include the relations of the structure; we present the details of this construction in Subsection 3.2. Before that, we show in Subsection 3.1 how assignments can be encoded in
terms of tree-decompositions. Decomposition automata constitute a special kind of automata that compute assignments; more precisely, they run over tree-representations of structures, and produce assignments encoded over tree-decompositions. We prove, in Subsection 3.3, that for any MSO formula \( \phi \) there exists a decomposition automaton producing the satisfying assignments of \( \phi \) over any decomposed structure of bounded treewidth. Thus, solving the evaluation problem for decomposition-automata we directly obtain the solution to the initial MSO evaluation problem.

### 3.1 Encoding assignments over tree-decompositions

Let us now see how assignment automata [14], which run over trees and return tuples over subsets of the tree domain, can be generalized in the case of tree-decompositions. The main issue of this generalization is the definition of a proper encoding of assignments in the form of a total mapping with domain the nodes of the tree-decomposition.

Recall from [14] that in the case of trees we used to encode an assignment \( \overline{B} = (B_1, \ldots, B_k) \in \mathcal{P}(T)^k \) via a total mapping \( \varepsilon \) from the set \( T \) of nodes of the input tree to the set \( \{0, 1\}^k \) s.t. \( \varepsilon(n) = (e_1, \ldots, e_k) \) with \( e_i = 1 \) iff \( n \in B_i \), \( 1 \leq i \leq n \); i.e., \( 1 \) at position \( i \) of \( \varepsilon(n) \) indicates the participation of node \( n \) in set \( B_i \) of \( \overline{B} \).

A tree-decomposition can be thought as a special kind of tree having as nodes its ordered bags i.e. each node is of the form \((a_1, \ldots, a_\ell) \in A^\ell\), \( 1 \leq \ell \leq w + 1 \). We show that we can define a way to encode an assignment \( \overline{B} = (B_1, \ldots, B_k) \in (\mathcal{P}(A))^k \) via a total mapping \( \varepsilon \) by adapting the above encoding idea in this framework. There are two main differences in the case of tree-decomposition trees that we have to deal properly: i) we now have more than one element at a node and thus 0, 1 do not suffice to encode each \( B_i \) and ii) each element may occur at several nodes. As shown in the formal definitions given below, the value of this new encoding mapping \( \varepsilon \) at a node \( n \) of the tree-decomposition corresponds to the union of values \( \varepsilon(a) \) for all \( a \) in the bag of \( n \); more precisely, in order to be able to distinguish to which \( a \) the containment in \( B_i \) refers to, we have to use the set \( W = \{1, \ldots, w + 1\} \) of positions of the tree-decomposition bags instead of set \( \{0, 1\} \). We overcome (ii) by choosing one representative position for each element, namely the position of its topmost occurrence in the tree-decomposition tree.

**Definition 3.1.** Let \( \mathcal{I} = (\mathcal{T}, (\overline{a}_n)_{n \in \mathcal{T}}) \) be a special tree-decomposition of a finite structure with domain \( A \) and treewidth \( w \geq 1 \). Given \( \mathcal{I} \), every element \( a \in A \) is uniquely associated to a pair \( \text{pair}_\mathcal{I}(a) = (n, j) \in \mathcal{T} \times W \), \( W = \{1, \ldots, w + 1\} \), indicating the topmost occurrence of \( a \) in \( \mathcal{I} \) and called the (representative) pair of \( a \) w.r.t. \( \mathcal{I} \):

\[
\text{pair}_\mathcal{I}(a) = (n, j) \quad \text{iff} \quad a_n^j = a \quad \& \quad a_{n'}^i \neq a \quad \text{for} \quad n' \quad \text{the parent of} \quad n \quad \text{in} \quad \mathcal{T} \quad \text{and} \quad 1 \leq i \leq |a_n^j|;
\]

we shall call (representative) node of \( a \) w.r.t. \( \mathcal{I} \), denoted \( \text{node}_\mathcal{I}(a) \), the projection over the first position of \( \text{pair}_\mathcal{I}(a) \); that is, \( \text{node}_\mathcal{I}(a) = n \) iff there exists \( j \in W \) s.t. \( \text{pair}_\mathcal{I}(a) = (n, j) \). Moreover we define the set \( \text{Pos}_\mathcal{I}(n) \) of element positions at \( n \) w.r.t. \( \mathcal{I} \) as the set \( \{j \in W \mid \exists a \in A \quad (n, j) = \text{pair}_\mathcal{I}(a)) \} \) of positions of ordered bag \( \overline{a}_n \) where the elements of \( A \) having their topmost occurrence at \( n \) occur.

**Definition 3.2.** Let \( k, w \) be positive integers and let \( T, A \) be finite sets; let \( \mathcal{E}_{k, w}(T) \) be the set containing any total mapping \( \varepsilon : T \rightarrow (\mathcal{P}(W))^k \) where \( W = \{1, \ldots, w + 1\} \); and let \( \mathcal{D}_{k, w}(T, A) \) be the set of special tree-decompositions, with width \( w \) and node set \( T \), of structures having domain
A. We define the **encoding mapping** $\text{enc}_{k,w} : \bigcup_{A,T}(\mathcal{P}(A))^k \times \mathcal{D}_{\text{dec}}(T,A) \rightarrow \bigcup_{T}(\mathcal{E}_{k,w}(T))$ as follows: for $k$-ary assignment $\overline{B} = (B_1,\ldots,B_k) \in (\mathcal{P}(A))^k$ and $\mathcal{I} = (T, (\overline{a}_n)_{n \in T}) \in \mathcal{D}_{\text{dec}}(T,A)$, we have $\text{enc}_{k,w}(\overline{B}, \mathcal{I}) = \varepsilon_{\overline{B}}$ with $\varepsilon_{\overline{B}} : T \rightarrow (\mathcal{P}(W))^k$ such that for all $n \in T$,

$$
\varepsilon_{\overline{B}}(n) = (I^n_1, \ldots, I^n_k) \iff I^n_i = \{ j \in \text{Pos}_T(n) \mid a^n_j \in B_i \} \subseteq W, \ 1 \leq i \leq k
$$

i.e. each $I^n_i$ is the set of those element positions at $n$ that occupy elements of $B_i$. We call $\varepsilon_{\overline{B}}$ the **mapping encoding assignment** $\overline{B}$ (or the **encoding of** $\overline{B}$) with respect to tree-decomposition $\mathcal{I}$.

**Decoding mapping** $\text{dec}_{k,w} : \bigcup_{A,T}(\mathcal{E}_{k,w}(T) \times \mathcal{D}_{\text{dec}}(T,A)) \rightarrow \bigcup_A((\mathcal{P}(A))^k)$ is defined as follows: for $\varepsilon : T \rightarrow (\mathcal{P}(W))^k$ with $\varepsilon(n) = (I^n_1, \ldots, I^n_k)$, and tree-decomposition $\mathcal{I} = (T, (\overline{a}_n)_{n \in T}) \in \mathcal{D}_{\text{dec}}(T,A)$, we have $\text{dec}_{k,w}(\varepsilon, \mathcal{I}) = \overline{B}_\varepsilon$ with $\overline{B}_\varepsilon = (B_1, \ldots, B_k) \in (\mathcal{P}(A))^k$ such that $B_i = \bigcup_{n \in T} \{ a^n_j \mid j \in I^n_i \}, \ 1 \leq i \leq k$.

Let $\mathcal{I} = (T, (\overline{a}_n)_{n \in T}) \in \mathcal{D}_{\text{dec}}(T,A)$; each mapping $\varepsilon : T \rightarrow (\mathcal{P}(W))^k$ such that $\varepsilon(n) \in (\mathcal{P}(\text{Pos}_T(n)))^k$ for all $n \in T$ is called an assignment mapping with respect to $\mathcal{I}$.

**Example 3.1.** Consider the assignment $\overline{B} = (B_1, B_2)$ where $B_1 = \{ a, c \}$ and $B_2 = \{ b, h, c \}$ over the graph and its tree decomposition given in Figure 1, page 5; $\overline{B}$ is encoded by the assignment mapping such that $n_1 \mapsto (\{1\}, \emptyset), n_2 \mapsto (\emptyset, \{3\}), n_3 \mapsto (\emptyset, \{2\}), n_4 \mapsto (\emptyset, \{2\}), n_5 \mapsto (\{3\}, \emptyset)$.

**Remark 3.1.** Let $\mathcal{I} = (T, (\overline{a}_n)_{n \in T}) \in \mathcal{D}_{\text{dec}}(T,A)$; and let $\mathcal{E}_{k,w,I} \subseteq \mathcal{E}_{k,w}(T)$ be the set of assignment mappings with respect to $\mathcal{I}$; by definition, $\mathcal{E}_{k,w,I}$ is the image of the restriction of $\text{enc}_{k,w}$ to $(\mathcal{P}(A))^k \times \mathcal{I}$. Decoding mapping $\text{dec}_{k,w}$ restricted to $\mathcal{E}_{k,w,I} \times \mathcal{I}$ corresponds to the “inverse mapping” of the encoding mapping $\text{enc}_{k,w}$ restricted to $(\mathcal{P}(A))^k \times \mathcal{I}$ in the sense that for any $\varepsilon \in \mathcal{E}_{k,w,I}$, we have $\text{dec}_{k,w}(\varepsilon, \mathcal{I}) = (B_1, \ldots, B_k)$ iff $\text{enc}_{k,w}((B_1, \ldots, B_k), \mathcal{I}) = \varepsilon$.

Using the encoding mapping defined in this subsection, we shall define assignment-automata that run over extended tree-decompositions and produce as part of their runs mappings encoding assignments of MSO-formulas. These automata, called decomposition-automata, compute the satisfying assignments of MSO-formulas by decoding -via $\text{dec}_{k,w}$- the assignment mappings which are computed as parts of their successful runs.

### 3.2 Combining structures & tree-decomposition

In this subsection we define formally the trees which constitute the input trees for our automata. As we have already mentioned, decomposition-automata are tree-automata that run over colored trees encoding tree-decompositions and structures.

Let $\mathcal{A}$ be an input $\tau$-structure, $\tau = \{ R_1, \ldots, R_t \}$, and let $\mathcal{I} = (T, (\overline{a}_n)_{n \in T})$ be an ordered tree-decomposition of $\mathcal{A}$ having width $w$. A simple way to define pair $(\mathcal{A}, \mathcal{I})$ in a form of colored-tree is by properly extending the bags of the tree-decomposition in order to include the relations of an input structure. This is exactly the idea of the definition of tree $\mathcal{T}_{\mathcal{I}, \mathcal{A}}$ defined below.

More precisely, $\mathcal{T}_{\mathcal{I}, \mathcal{A}}$ is a colored-tree defined as follows: its underlying tree is the tree $\mathcal{T}$ of the tree-decomposition $\mathcal{I}$; and its coloring function $c$ encodes at each node $n$ the structural
properties of $\mathcal{I}$ (and in particular the size of the bag $a_n$ and the intersection of $a_n$ with the bag $a_{n'}$ of the parent $n'$ of $n$, see the first two positions of $c(n)$ below) and the isomorphism type of substructure $\mathcal{A}|_{a_n}$ of $\mathcal{A}$ induced by $a_n$ (last $\ell$ positions of $c(n)$ below). That is,

$$\mathcal{T}_{\mathcal{I},\mathcal{A}} = (\mathcal{T}, c),$$

where $c : T \to \Gamma_{\tau,w}, \Gamma_{\tau,w} = W \times \mathcal{P}(W^2) \times \mathcal{P}(W^{r_1}) \times \cdots \times \mathcal{P}(W^{r_\ell}), W = \{1, \ldots, w + 1\}$ with $c(n) = (s, P, S_1, \ldots, S_\ell)$ where

- $s = |a_n|$, 
- $P = \{\emptyset, \{(i, j) \mid a_n^i = a_{n'}^j\}\}$ if $n$ is the root;
- $P = \{(i, j) \mid a_n^i = a_{n'}^j\}$ if $n'$ is the parent of $n$.

- $S_i = \{(j_1, \ldots, j_{r_i}) \mid (a_n^{j_1}, \ldots, a_n^{j_{r_i}}) \in R_i^A\}, \quad 1 \leq i \leq \ell$.

We call the $\Gamma_{\tau,w}$-tree $\mathcal{T}_{\mathcal{I},\mathcal{A}}$ the tree of $\mathcal{A}$ with respect to $\mathcal{I}$.

Note that $\Gamma_{\tau,w}$-trees offer a uniform way to refer to trees encoding pairs $(\mathcal{A}, \mathcal{I})$ where $\mathcal{A}$ is any $\tau$-structure and $\mathcal{I}$ is a tree-decomposition of $\mathcal{A}$ of width $w$. Notice that not every $\Gamma_{\tau,w}$-tree represents a tree of a $\tau$-structure with respect to a tree-decomposition of width $w$; $w$-decomposition automata (of Definition 3.3) run over $\Gamma_{\tau,w}$-trees that encode pairs $(\mathcal{A}, \mathcal{I})$.

**Example 3.2.** Consider as input structure and tree-decomposition, the graph $\mathcal{G}$ and its corresponding tree-decomposition $\mathcal{I}$ of Figure 1; the definition of coloring function $c$ of tree $\mathcal{T}_{\mathcal{I},\mathcal{A}}$ is given in the following table:

<table>
<thead>
<tr>
<th>$n$</th>
<th>$c(n)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_1$</td>
<td>$(3, \emptyset, {(1, 2), (2, 3), (2, 1), (3, 2)})$</td>
</tr>
<tr>
<td>$n_2$</td>
<td>$(3, {(1, 1), (2, 3)}, {(1, 3), (2, 3), (3, 1), (3, 2)})$</td>
</tr>
<tr>
<td>$n_3$</td>
<td>$(2, {(1, 2)}, {(1, 2), (2, 1)})$</td>
</tr>
<tr>
<td>$n_4$</td>
<td>$(3, {(1, 1), (3, 3)}, {(1, 2), (1, 3), (2, 3), (2, 1), (3, 1), (3, 2)})$</td>
</tr>
<tr>
<td>$n_5$</td>
<td>$(3, {(1, 3)}, {(1, 2), (1, 3), (2, 3), (2, 1), (3, 1), (3, 2)})$</td>
</tr>
</tbody>
</table>

**Remark 3.2.** The definition of $\mathcal{T}_{\mathcal{I},\mathcal{A}}$ resembles the definition of $\mathcal{T}^*$ of [12], but $\mathcal{T}^*$ is defined with respect to ordered tree-decompositions whose bags $\overline{a}_n$ have constant length, namely $(w + 1)$. Since the constant length is achieved by allowing repeated occurrences of elements of $a_n$ in $\overline{a}_n$, a different component, that denotes the positions of $\overline{a}_n$ where identical elements occur, replaces the first component of our colors. More precisely, $\mathcal{T}^*$ is a binary $\Gamma_{\tau,w}^*$-colored tree, $\Gamma_{\tau,w}^* = \mathcal{P}(W^2) \times \mathcal{P}(W^2) \times \mathcal{P}(W^{r_1}) \times \cdots \times \mathcal{P}(W^{r_\ell})$. Although the definition of coloring function of these two trees is almost the same, $\mathcal{T}_{\mathcal{I},\mathcal{A}}$ constitutes a simplification of the definition of $\mathcal{T}^*$ with
respect to an arbitrary binary tree-decomposition \(I\): i) we do not have to stuff the bags with “useless” repetitions of elements in order to have the same lengths for all ordered bags \(\pi_n\)'s and also encode them in the coloring function; ii) the repetitions result in multiple representations of the same information in sets \(P\) and \(S_i\)'s of \(\gamma\) (e.g. suppose that \(a_n = (a, b, a, b)\); for \((a, b) \in R_i^A\) we have \((1, 2), (1, 4), (3, 2), (3, 4) \in S_i\) w.r.t. \(T^*\)); that is, we have a significant pointless increase in the size of the colors. Moreover, the repetitions of the same element in an ordered bags result in more complex definitions of the representative pair of an element and thus of the automata.

### 3.3 A direct reduction of the MSO-evaluation problem to an automata problem: \(w\)-decomposition automata

**\(w\)-Decomposition assignment automata for MSO formulas: \(w\text{-dec-Assign}_\phi\).** We define below decomposition-automata for MSO[\(\tau\)] formulas. Decomposition-automata, generalizing assignment-automata defined in [14], have more involved automata definitions and computation mechanisms which are defined with respect to both the initial structure and the corresponding \(\tau\)-nodes of \(w\)-Decomposition assignment automata for MSO formulas:

\[A_{\phi,w} = (\Gamma_{\tau,w}, Q_{k,w}, \Delta_0, \Delta, F_{k,w})\]

is a special kind of non-deterministic bottom-up tree automata: \(Q_{k,w} = Q \times (\mathcal{P}(W))^k\), \(W = \{1, \ldots, w+1\}\) is the set of states and \(F_{k,w} = F \times (\mathcal{P}(W))^k\) is the set of final states; for finite set \(Q\) and \(F \subseteq Q\); i.e. a state is a \((k+1)\)-tuple \((q, I_1, \ldots, I_k)\) (abbreviated as \(qT\)); \(\Delta_0 \subseteq \Gamma_{\tau,w} \times Q_{k,w}\) is the start transition relation and \(\Delta \subseteq Q \times Q \times \Gamma_{\tau,w} \times Q_{k,w}\) is the transition relation. Decoding mapping \(\text{dec}_{k,w}\), introduced in Definition 3.2, describes the computation mechanism of \(w\text{-dec-Assign}_\phi\) consisting in the decoding of runs into assignments with the help of a tree-decomposition.

The run \(\rho\) of \(w\text{-dec-Assign}_\phi\) over \(\Gamma_{\tau,w}\)-colored tree \(T_{I,A}\) is a mapping assigning states to nodes of \(T_{I,A}\) such that

- if \(n\) is a leaf with color \(\gamma\), then \(\rho(n) = qT\) if there exists a transition \((\gamma, qT) \in \Delta_0\), and
- if \(n\) is a node of color \(\gamma\), having children \(n_1, n_2\), then the value \(\rho(n)\) is such that \(\rho(n_1) = qT\), \(\rho(n_2) = \bar{p}T\) and \((q, p, \gamma, \rho(n)) \in \Delta\).

A run is **successful** if it maps the root to a final state.

Notice that a run \(\rho : T \rightarrow Q \times (\mathcal{P}(W))^k\) of \(w\text{-dec-Assign}_\phi\) can be thought as the concatenation \(\varsigma; \varepsilon\) where \(\varsigma : T \rightarrow Q\) and \(\varepsilon : T \rightarrow (\mathcal{P}(W))^k\) s.t. \(\rho(n) = (q, \bar{T})\) iff \(\varsigma(n) = q\) and \(\varepsilon(n) = \bar{T}\); \(\varsigma, \varepsilon\) are called the **state part** and **assignment part** of run \(\rho\) respectively.

We say that \(w\text{-dec-Assign}_\phi\) computes the assignment \(B = (B_1, \ldots, B_k) \in (\mathcal{P}(A))^k\) iff there exists a successful run \(\varsigma; \varepsilon\) of \(w\text{-dec-Assign}_\phi\) over \(T_{I,A}\) such that \(\text{dec}_{k,w}(\varepsilon, I) = (B_1, \ldots, B_k)\). The set of assignments computed by \(w\text{-dec-Assign}_\phi\) over \(T_{I,A}\) is denoted \(w\text{-dec-Assign}_\phi(T_{I,A})\).

Note that each mapping \(\varepsilon\) that is the assignment part of a successful run of \(w\text{-dec-Assign}_\phi\) is an assignment mapping (recall Definition 3.2), and there exists a one-to-one correspondence between assignments computed via \(w\text{-dec-Assign}_\phi\) and its successful runs.

Notice that for sake of simplicity the above given definition of decomposition automata concerns MSO-formulas with \(k\) second-order variables i.e. formulas \(\phi(X_1, \ldots, X_k)\); the definition
is easily adapted for the general case of formulas \( \phi(x_1, \ldots, x_u, X_1, \ldots, X_v) \), \( k = u + v \), extending naturally the above. More precisely, it suffices to replace decoding mapping \( \text{dec}_{k,w} \) by a new decoding mapping \( \text{dec}_{u,v,w} \) defined as follows: \( \text{dec}_{u,v,w}(\varepsilon, I) = (a_1, \ldots, a_u, B_1, \ldots, B_v) \) iff \( \text{dec}_{k,w}(\varepsilon, I) = \{(a_1), \ldots, \{a_u\}, B_1, \ldots, B_v\} \) for all \( \varepsilon \) being the assignment parts of successful runs of \( \text{w-dec-Assn}_\phi \) over \( T_{I,A} \).

We define below the counting version of decomposition automata which we use to solve MSO-definable optimization problems.

**Definition 3.4.** Let \( \phi(X) \) be an MSO[\( \tau \)] formula; a counting \( w \)-decomposition automaton for \( \phi \), denoted \( w \text{-dec-Assn}^{\text{count}}_\phi \), is defined via \( w \text{-dec-Assn}_\phi \) differing from the latter only in the run definition: the run \( \rho \) of \( w \text{-dec-Assn}^{\text{count}}_\phi \) over \( T_{I,A} \) is a mapping \( T \rightarrow Q \times P(W) \times \{1, \ldots, |A|\} \) such that

- if \( n \) is a leaf with color \( \gamma \), then \( \rho(n) = (qI, |I|) \) if there exists a transition \( (\gamma, qI) \in \Delta_0 \).
- if \( n \) is a node of color \( \gamma \), having children \( n_1, n_2 \) with \( \rho(n_1) = (rJ, j) \), \( \rho(n_2) = (pL, \ell) \) and \( (r,p,\gamma, qI) \in \Delta \), then \( \rho(n) = (qI, j + \ell + |I|) \).

**Main theorem: the MSO-automata connection via decomposition automata.** Theorem 3.1, presented below, states that for every MSO[\( \tau \)]-formula \( \phi \) there exists a decomposition automaton for \( \phi \) that runs over trees of \( \tau \)-structures and computes the satisfying assignments of \( \phi \) over the \( \tau \)-structures.

The proof that we give is constructive, performed via induction on the structure of MSO formula \( \phi \). In fact, what we prove is the existence of a deterministic bottom-up tree-automaton \( w \text{-dec-Assn}_\phi \) accepting \( \Gamma_{\tau,w} \)-trees encoding - via \( \text{enc}_{k,w} \) - the satisfying assignments of formula \( \phi \). The connection of \( w \text{-dec-Assn}_\phi \) and \( w \text{-dec-Assn}^{\text{count}}_\phi \) is the following: \( \varsigma; \varepsilon_B \) is a run of \( w \text{-dec-Assn}_\phi \) over \( \Gamma_{\tau,w} \)-tree \( T_{I,A} = (T, \varsigma) \) iff \( \varsigma \) is a run of \( w \text{-dec-Assn}^{\text{count}}_\phi \) over \( \Gamma_{\tau,w} \times (P(W))^k \)-tree \( (T_{I,A}; \varepsilon_B) = (T, c; \varepsilon_B) \) extending \( T_{I,A} \) by satisfying assignment \( \varepsilon_B \). More precisely, let \( w \text{-dec-Assn}_\phi = (\Gamma_{\tau,w} \times (P(W))^k, Q, w_{\Delta_0}, w_{\Delta}, F) \); the transition relations \( w_{\Delta_0}, w_{\Delta} \) of automaton \( w \text{-dec-Assn}^{\text{count}}_\phi = (\Gamma_{\tau,w} \times (P(W))^k, w_{\Delta_0}, w_{\Delta}, F \times (P(W))^k) \) can be easily derived via the transitions functions \( w_{\Delta_0}, w_{\Delta} \) of \( w \text{-dec-Assn}_\phi \) as follows: i) \( (\gamma, (q, T)) \in w_{\Delta_0} \) iff \( w_{\Delta_0}((\gamma, T)) = q \) and ii) for all \( T_1, T_2 \in (P(W))^k \), \( (\gamma_1, T_1), (\gamma_2, T_2) \) \( (\gamma, T) \in w_{\Delta} \) iff \( w_{\Delta}(\gamma_1, q_2, (\gamma, T)) = q \). Thus, we often use \( w \text{-dec-Assn}_\phi \) as concise representation of \( w \text{-dec-Assn}^{\text{count}}_\phi \). Notice that another reason for this representation choice is the fact that the runs of \( w \text{-dec-Assn}_\phi \) can be directly computed via the transitions functions \( w_{\Delta_0}, w_{\Delta} \) of \( w \text{-dec-Assn}_\phi \).

Notice that, due to encodings via \( \text{enc}_{k,w} \), each satisfying assignment is produced via a unique run of \( w \text{-dec-Assn}_\phi \).

Before we present the main theorem, we give a lemma that is used in its proof. Recall that in a tree \( T \) a node \( n \) is in a lower level that some other node \( n' \), denoted \( n \prec_T n' \), if the distance of \( n \) from the root is greater than the distance of \( n' \) from the root. Moreover, let \( \overline{T}_n = \{a \in A \mid \text{node} \_\text{set}(a) \in T \setminus T_n\} \).

**Lemma 3.1.** Let \( A \) be a \( \tau \)-structure with domain \( A \) and let \( I \) be a tree-decomposition of \( A \). Consider \( R \in \tau \) of arity 2; we say that \( b \) is an \( R^A \)-match of \( a \) if either \( (a, b) \in R^A \) or \( (b, a) \in R^A \).

Suppose that for \( a \in A \), \( \text{node} \_\text{set}(a) = n \); the following holds: the \( R^A \)-matches of \( a \) that are contained in \( \overline{A}_n \) are all also contained in bag \( a_n \) of \( n \).
**Proof.** The result is immediate consequence of properties (i) and (ii) of tree-decomposition definition. Suppose that \((b,c) \in R^A\) and that \(\text{node}_T(b) = n\) and \(\text{node}_T(c) = n'\); it suffices to prove that if \(n <_T n'\), then \(\{b,c\} \in a_n\). It follows by (ii) of tree-decomposition definition that since \((b,c) \in R^A\) there exists a node \(v\) such that \(\{b,c\} \in a_v\). Since \(\text{node}_T(b) = n\), i.e. \(b\) has its topmost occurrence in \(a_n\), either \(v = n\) or \(v <_T n\). Suppose that \(v <_T n\); since \(c \in a_v\), and \(c \in a_{v'}\), and \(n <_T n'\), by (i) of tree-decomposition definition we moreover have that \(c \in a_n\). □

**Theorem 3.1.** For every \(k\)-ary MSO[\(\tau\)]-formula \(\phi\) and for every \(w \geq 1\) there exists a decomposition automaton \(w{-}\text{dec-Assign}_\phi\) that, for any \(\Gamma_{\tau,w}\)-colored tree \(T_{\ell,A}\) encoding pair \((A,I)\) where \(A\) is a \(\tau\)-structure of treewidth at most \(w\) and \(I\) is a special tree-decomposition of \(A\), computes the satisfying assignments of \(\phi\) over \(A\) i.e. \(\phi(A) = w{-}\text{dec-Assign}_\phi(T_{\ell,A})\).

**Proof.** The proof is done via induction on the structure of MSO formula \(\phi\). As we have already mentioned, instead of actually constructing decomposition automaton \(w{-}\text{dec-Assign}_\phi\) we construct the deterministic automaton \(w{-}\text{dec-A}_\phi\). Recall from above that the latter constitutes a concise representation of the former. Moreover, the involved definitions and proofs become more natural in the context of a deterministic automaton as \(w{-}\text{dec-A}_\phi\) is.

The deterministic automaton \(w{-}\text{dec-A}_\phi\), that we construct, accepts the class of \((\Gamma_{\tau,w} \times (\mathcal{P}(\mathcal{W}))^k)\)-trees encoding the satisfying assignments of \(\phi\) over \(\tau\)-structures having treewidth at most \(w\). In the sequel of the proof, we shall also refer to such trees as trees extended with assignments or assignment mappings since they are in fact derived from \(\Gamma_{\tau,w}\)-trees by extending their coloring functions with assignment mappings encoding the satisfying assignment of \(\phi\).

Notice that the transitions of \(w{-}\text{dec-A}_\phi\) have the form \(q_1,q_2,(\gamma,\overline{T}) \rightarrow q\).

We consider first the case of atomic formulas: the automata for \(\text{In}(x,X)\) and \(\text{R}(x,y)\) (i.e. \(R_i = R\) with \(r_i = 2\)) are given in Table 3 and Table 4 respectively. The conditions used in their definitions are given in Table 1 whereas the meaning of the states for the case of the automaton for \(\text{R}(x,y)\) are given in Table 2.

Notice that, the transitions’ tables (Table 3 and Table 4) -constituting a concise representation of transitions- are read as follows: every row (different from the last one) containing transition \(q_1,q_2,(\gamma,\overline{T}) \rightarrow q\) with condition \(c(\overline{T})\) corresponds to the following two sets of transitions: i) set \(\{q_1,q_2,(g,\overline{T}) \rightarrow q \mid \overline{T}\) has the same form as \(\overline{T} \& g \in \Gamma_{\tau,w}\) satisfies condition \(c(\overline{T})\}\) of transitions with colors satisfying the involved condition and ii) set \(\{q_1,q_2,(g,\overline{T}) \rightarrow qf \mid \overline{T}\) has the same form as \(\overline{T} \& g \in \Gamma_{\tau,w}\) does not satisfies condition \(c(\overline{T})\}\) of transitions with colors not satisfying the involved condition (derived by the last row of the tables); the last row of the table covers also all cases of transitions with some of \(q_1, q_2\) or \(\overline{T}\) not having been considered in previous rows. Notice that the notion of same form of two \(k\)-ary vectors \(\overline{T}\) and \(\overline{T}\) is w.r.t. occurrences of emptyset and singletons in the same positions. Start transitions are derived from the transitions of the form \(q_0,q_0,(\gamma,\overline{T}) \rightarrow q\) i.e. for every such transition there exists a start transition of the form \((\gamma,\overline{T}) \rightarrow q\).

It is not difficult to verify the correctness of the automata constructions for atomic formulas. We give the proof for \(\phi = R(x,y)\) where \(R\) corresponds to a symmetric relation: we first prove that each satisfying assignment of \(R\) is computed via a successful run of \(w{-}\text{dec-Assign}_R\) (**); then we prove that if \(\epsilon\) is the assignment part of a successful run of \(w{-}\text{dec-Assign}_R\) then \(\epsilon\) encodes a satisfying assignment of \(R\) (**).
Table 1: Conditions/Sets useful in the transitions definitions.

<table>
<thead>
<tr>
<th>Conditions/Sets</th>
<th>w.r.t. $\gamma \in \Gamma_{\tau,w}$</th>
<th>w.r.t. $\mathcal{I}, \mathcal{A}$ at $n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$element(j)$</td>
<td>$j \leq \gamma_1$ &amp; $\forall i \in W ((j,i) \notin \gamma_2)$</td>
<td>$j \leq</td>
</tr>
<tr>
<td>$set(I)$</td>
<td>$\forall i \in I (element(i))$</td>
<td>$\forall i \in W (a^i_n)$</td>
</tr>
<tr>
<td>$SameInPar(J)$</td>
<td>${i \mid \exists j \in J ((j,i) \in \gamma_2)}$</td>
<td>${i \mid \exists j \in J (a^j_n = a^i_n)}$</td>
</tr>
<tr>
<td>$R(j)$</td>
<td>${i \mid (j,i) \in \gamma_3}$</td>
<td>${i \mid (a^j_n, a^i_n) \in R^4}$</td>
</tr>
<tr>
<td>$RInPar(j)$</td>
<td>$SameInPar(R(j))$</td>
<td>${i \mid (a^j_n, a^i_n) \in R^4}$</td>
</tr>
</tbody>
</table>

Table 2: The meaning of states of $w$-dec-$\text{Assign}_{R(x,y)}$.

<table>
<thead>
<tr>
<th>State</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>no node has been selected yet</td>
</tr>
<tr>
<td>$q_1$</td>
<td>an element $a$ (resp. $b$) has already been selected &amp; the set ${a^i_p \mid i \in I}$, (resp. $q_2$)</td>
</tr>
<tr>
<td>$a_p$</td>
<td>where $p$ is the father of the current node $n$, contains all nodes $b$ (resp. $a$) in $\overline{A}_n$ s.t. $(a,b)$ is a sat. assignment of $R(x,y)$</td>
</tr>
<tr>
<td>$q_{af}$</td>
<td>final state: a satisfying assignment has been selected</td>
</tr>
<tr>
<td>$q_f$</td>
<td>failure of acceptance: a non satisfying assignment has been selected</td>
</tr>
</tbody>
</table>

For the proof of (*), suppose that $(b,c)$ is a satisfying assignment of $R$ over an input structure $\mathcal{A}$ with tree-decomposition $\mathcal{I}$; and suppose that $n = node_{\mathcal{I}}(b)$ and $n' = node_{\mathcal{I}}(c)$. We distinguish the following cases: i) $n = n'$, ii) $n < n'$, (the case that $n > n'$ is completely symmetric). In case (i), $(b,c) \in R^4$ is computed by $w$-dec-$\text{Assign}_R$ via unique successful run $\rho$ such that $\rho(n) = (q_0, (\{i\}, \{j\}))$ where $i,j$ are the positions of $\pi_n$ where $b,c$ occur (condition $R(i,j)$ is satisfied at $n$): $\rho(n') = (q_0, (\emptyset, \emptyset))$ for all $n' > n$; and $\rho(n') = (q_0, (\emptyset, \emptyset))$ for all $n' < n$. In case (ii), $w$-dec-$\text{Assign}_R$ -proceeding bottom-up- computes $(b,c) \in R^4$ via successful run $\rho$ as follows: $\rho(v) = (q_0, (\emptyset, \emptyset))$ for all nodes $v < n$; at node $n$, $\rho$ “selects” $b$ assigning state $(q_1, (\{i\}, \emptyset))$ where $i$ is the position of bag $a_n$ where $b$ occurs, and $I$ contains the positions of the bag $a_p$ of the parent of $n$ where the $R$-matches of $b$, that are contained in $a_n \cap a_p$, occur (condition $element(i)$ & $RInPar(i) = I \neq \emptyset$ is satisfied at $n$). By Lemma 3.1, the $R$-matches of $b$ that are contained in $a_n \cap a_p$ are exactly the $R$-matches of $b$ contained in $\overline{A}_n$ i.e. $\{d \in \overline{A}_n \mid (b,d) \in R^4\} = \{a^i_p \mid i \in I\}$; thus $c \in \{a^i_p \mid i \in I\}$. Then, for all nodes $v$ in the path connecting $n$ and $n'$ we have $\rho(v) = (q_1, (\emptyset, \emptyset))$ where $J \subseteq W$ is such that $\{d \in \overline{A}_v \mid (b,d) \in R^4\} = \{a^i_p \mid i \in J\}$ and $c \in \{a^i_p \mid i \in J\}$, $a_p$ denotes the bag of the parent of $v$ (condition $SameInPar(I) = J \neq \emptyset$ is satisfied at each $v$ when $(q_1, (\emptyset, \emptyset))$ is the state assigned by $\rho$ at the child of $v$ in the path between $n$ and $n'$). At $n'$, we have $\rho(n') = (q_0, (\emptyset, \{j\}))$ where
Table 3: Transitions of \( w\text{-dec-Ass}ign_{\text{in}(x,X)} \):

<table>
<thead>
<tr>
<th>Transition</th>
<th>Conditions satisfied by ( \gamma \in \Gamma_{\tau_\gamma,w} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0, q_0, \gamma, (\emptyset, \emptyset) ) &amp; ( q_0 )</td>
<td></td>
</tr>
<tr>
<td>( q_0, q_0, \gamma, (\emptyset, I) ) &amp; ( q_0 ) set(I)</td>
<td></td>
</tr>
<tr>
<td>( q_0, q_0, \gamma, {i}, I ) &amp; ( q_a ) set(I) &amp; ( i \in I )</td>
<td></td>
</tr>
<tr>
<td>( q_0, q_a, \gamma, (\emptyset, \emptyset) ) &amp; ( q_a )</td>
<td></td>
</tr>
<tr>
<td>( q_0, q_0, \gamma, (\emptyset, I) ) &amp; ( q_a ) set(I)</td>
<td></td>
</tr>
<tr>
<td>( q_0, q_0, \gamma, (\emptyset, I) ) &amp; ( q_f ) all other cases</td>
<td></td>
</tr>
</tbody>
</table>

\( j \) is the position of \( a_{n'} \) where \( c \) occurs (condition element\((j) \& j \in I \) is satisfied at \( n' \) when \( (q^j_1, (\emptyset, \emptyset)) \) is the state assigned by \( \rho \) at the child of \( v \) in the path between \( n \) and \( n' \). We have \( \rho(v) = (q_a, (\emptyset, \emptyset)) \) for all \( v > n' \) and \( \rho(v) = (q_0, (\emptyset, \emptyset)) \) for all \( v \in T_u \) where \( u \) is the child of \( n' \) not in the path connecting it with \( n \). Thus, we have proven that each satisfying assignment is computed via a (unique) successful run.

For (**), we have to prove that the successful runs of Assign-dec-\( w_R \) corresponds exactly to the satisfying assignments \( \in R^A \). It follows from the definition of Assign-dec-\( w_R \) that the form of successful runs have exactly the form of runs defined in cases (i) and (ii) above, encoding indeed only satisfying assignments.

Notice that we gave the proof for the case of a binary symmetric relation for sake of simplicity. The construction is easily generalized. For example, in the case where \( R \) does not correspond to a symmetric relation we obtain the corresponding automaton via the following slight modification of the definition of Assign-dec-\( w_R \): we just replace the second block of transitions of Table 4 with transitions \( q_0, q_0, \gamma, \{j\}, \emptyset \) & \( q^I_0 \) when \( \gamma \) satisfies element\((j) \& R_2\text{InPar}(j) = I \neq \emptyset \) and \( q_0, q_0, \gamma, (\emptyset, \{j\}) \) & \( q^1_0 \) when \( \gamma \) satisfies element\((j) \& R_1\text{InPar}(j) = I \neq \emptyset \); \( R_1(j) = \{i \mid (i, j) \in \gamma_3 \} \), \( R_2(j) = \{i \mid (j, i) \in \gamma_3 \} \) and \( R_i\text{InPar}(j) = \text{SameInPar}(R_i(j)) \) for \( i = 1, 2 \). Notice also that the decomposition-automaton for a \( k \)-ary atomic formula \( P(x_1, \ldots, x_k) \) constitutes a generalization of the decomposition-automaton for the binary atomic formula \( R(x, y) \). More precisely, the states of Assign-dec-\( w_P \) should be \( q_0, q_a, q_f \) and \( q^{i_1, \ldots, i_k} \), \( (I_1, \ldots, I_k) \in (\mathcal{P}(W))^k \).

Suppose that \( \phi \) is a non-atomic formula. The definition of automata \( \text{w-dec-A}_\phi \) for the cases where \( \phi = \phi_1 \wedge \phi_2 \) and \( \phi = \exists X \psi \) follows the classic deterministic tree-automata inductive definitions i.e. in the first case the automaton corresponds to the intersection of \( \text{w-dec-A}_{\phi_1} \) and \( \text{w-dec-A}_{\phi_2} \) whereas in the second case the automaton is constructing via a proper projection over \( \text{w-dec-A}_{\phi} \) and a further determinization of the resulting automaton.

The only case that needs a special treatment is that of \( \phi = \neg \psi \). The standard construction defines \( A_\phi \) as the complement of \( A_\psi \) denoted \( A_\psi^{\text{compl}} \). Due to the fact that our automata accept satisfying assignments encoded in a specific way, the complement does not yield a correct construction in our case. More precisely, recall that automaton \( \text{w-dec-A}_\psi \) accepts all trees extended with the encodings of the satisfying assignments of \( \phi \) (i.e. in the form of assignment
mappings defined in Definition 3.2 based on representative pair notion); that is, it rejects those
trees that are extended via non assignment mappings among which there exist mappings whose
decoding via \( w\text{-dec-A}_\phi \) is a satisfying assignment. The latter trees are accepted via \( w\text{-dec-A}_\psi^{\text{compl}} \)
and thus \( w\text{-dec-Assign}_\psi^{\text{compl}} \) would compute assignments satisfying \( \psi \). We can eliminate such
productions by imposing on \( w\text{-dec-A}_\psi^{\text{compl}} \) the requirement to accept only assignment mappings;
the resulting automaton should be the correct automaton \( w\text{-dec-A}_\phi \). Thus, for \( \psi(x_1, \ldots, x_k) \)
we define \( w\text{-dec-A}_\phi \) as the intersection of automata \( w\text{-dec-A}_\psi \) and \( w\text{-dec-A}_{\text{Set}^k} \) where the latter
automaton is an automaton accepting only trees extended with the encodings of assignments
of the form \((B_1, \ldots, B_k)\) via \( \text{enc}_{k,w} \) (recall Definition 3.2). Analogously when \( \psi \) has the form
\( \psi(x_1, \ldots, x_n, X_1, \ldots, X_p) \), \( w\text{-dec-A}_\phi \) is the intersection of automata \( w\text{-dec-A}_\psi \), \( w\text{-dec-A}_{\text{Element}^u} \)
and \( w\text{-dec-A}_{\text{Set}^k} \); \( w\text{-dec-A}_{\text{Element}^u} \) is the automaton accepting only trees extended with the
encodings of assignments of the form \( (\{b_1\}, \ldots, \{b_n\}) \).

Optimized algorithms for the evaluation of decomposition automata over decomposed
structures are given via datalog in the next section. The following representations of \( w\text{-dec-Assign}_\phi \),
based on transition functions \( w\text{-}\delta_0, w\text{-}\delta \) of deterministic \( w\text{-dec-A}_\phi \), are used in the definition of
datalog programs. The computation of sizes of the defined transitions sets is performed via their
connection with \( w\text{-}\delta_0, w\text{-}\delta \) of \( w\text{-dec-A}_\phi \).

**Definition 3.5.** Let \( \phi \) be an MSO[\( \tau \)] formula with \( k \) free variables (over \( \tau \)-structures of bounded
treewidth); and let \( w\text{-dec-Assign}_\phi = (\Gamma_{\tau,w}, Q \times (\mathcal{P}(W))^k, w\text{-}\Delta_0, w\text{-}\Delta, F \times (\mathcal{P}(W))^k) \); we call \( \phi \)-
states the elements of set \( Q \) and we call \( w\text{-}\text{decomposition} \ \phi \text{-transitions} \) the elements of sets \( w\text{-D}_0, \)

<table>
<thead>
<tr>
<th>Transition</th>
<th>Conditions satisfied</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0, q_0, \gamma, (\emptyset, \emptyset) )</td>
<td>( \Rightarrow q_0 )</td>
</tr>
<tr>
<td>( q_0, q_0, \gamma, ({j}, \emptyset) )</td>
<td>( \Rightarrow q_1^j ) ( \text{element}(j) \land \text{RInPar}(j) = I \neq \emptyset )</td>
</tr>
<tr>
<td>( q_0, q_0, \gamma, (\emptyset, {j}) )</td>
<td>( \Rightarrow q_2^j ) ( \text{SameInPar}(J) = I \neq \emptyset )</td>
</tr>
<tr>
<td>( q_0, q_0, \gamma, (\emptyset, {j}) )</td>
<td>( \Rightarrow q_a ) ( \text{element}(j) \land j \in I )</td>
</tr>
<tr>
<td>( q_0, q_a, \gamma, (\emptyset, \emptyset) )</td>
<td>( \Rightarrow q_a )</td>
</tr>
<tr>
<td>( q_1, q_2, \gamma, I )</td>
<td>( \Rightarrow q_f ) all other cases</td>
</tr>
</tbody>
</table>
w-D′₀, w-D, w-D′ defined as follows
- w-D₀ = {γqI | (γ, (q, T)) ∈ w-Δ₀} (of size |Γτ,w| · 2^{k(w+1)});
- w-D₀′ = {γq | there exists T s.t. γqT ∈ w-D₀} of size at most |Γτ,w| · 2^{k(w+1)};
- w-D = {γqq₁q₂T | there exist T₁, T₂ s.t. ((q₁, T₁), (q₂, T₂), γ, (q, T)) ∈ w-Δ} (of size |Γτ,w| · |Q|² · 2^{k(w+1)});
- w-D′ = {γqq₁q₂ | there exists T s.t. γqq₁q₂T ∈ w-D} of size at most |Γτ,w| · |Q|² · 2^{k(w+1)}.

4 The datalog approach to MSO evaluation

4.1 From MSO to datalog: the idea

In this section we reduce the MSO evaluation problem over structures of treewidth to a datalog evaluation problem, generalizing the datalog solution of [14] for the MSO evaluation problem over trees. This reduction is based on the reduction of the initial MSO evaluation problem to a decomposition-automata evaluation problem. In fact, as we also did in the case of trees, the proposed datalog programs solve this reduced automata problem.⁴

Since in both cases, i.e. over trees and over structures of bounded treewidth, our datalog programs evaluate tree-automata, the main core of the corresponding datalog solutions is in essence the same. However, this generalization should not be considered as a trivial one, in the sense that we had to develop the decomposition-automata theory in order to be able to apply these same similar datalog programs that evaluate assignment automata. In fact we defined decomposition automata in this way having this uniform datalog solution in mind. Notice also that it is due to decomposition automata that we are able to obtain interesting datalog expressivity results.

Moreover, apart from the similarities of the datalog solutions in the cases of trees and in the case structures of bounded tree-width, there exist significant differences with the second case being more complex. More precisely, the datalog programs that we define here are defined over two different databases; the one consists of facts encoding an input τ-structure A and the other of facts encoding a special tree-decomposition I of A with width w. Moreover, these datalog programs, implementing the more complex decoding mapping decₜ,w, compute the satisfying assignments of a given MSO formula φ in a more involved way requiring the participation of the database of the input tree-decomposition.

The main stages of the datalog approach are summarized as follows:

- Using as input database facts corresponding to given structure A and given tree decomposition I (see Definition 4.1), we write rules producing facts describing the coloring function of tree Tₜ,A encoding A and I; these rules constitute program Πτ,w of Lemma 4.1. (Note that this stage does not exist in the case of the datalog solution over trees, where the colors of the input tree are part of the input database.)
- Then, using the transitions of w-dec-Assnₜ we write datalog rules that correspond to runs of w-dec-Assnₜ over Tₜ,A.

The datalog approach of [14], solving the MSO evaluation problem on finite binary trees, is summarized as follows: for a given formula φ, we defined datalog programs that -using the transitions of assignment automaton Assnₜ which are produced by a datalog program- compute either runs of Assnₜ on an input tree or directly the assignments that this automaton computes. The input database of those programs constitutes an encoding the input tree T.
- Finally, combining properly the produced information concerning runs of \( w \)-dec-Assign_\( \phi \) over \( T_{I,A} \) (e.g. successful states in the case of unary MSO queries) and facts describing the tree decomposition \( I \), we finally produce set \( \phi(A) \).

### 4.2 Producing trees of structures via input databases

**Definition 4.1.** We define the database \( D_A \) of a structure \( A \) of vocabulary \( \tau = \{ R_1, \ldots, R_\ell \} \) where each \( R_i \) has arity \( r_i \) in the natural way:

- \( r_i(a_1, \ldots, a_{r_i}) \in D_A \) iff \( A \models R_i(a_1, \ldots, a_{r_i}) \), \( 1 \leq i \leq \ell \).

Notice that \( D_A \) is a database over the domain \( A \) of \( A \).

Let \( I = (T_n, (\pi_n)_{n \in T}) \) be a special tree-decomposition of width \( w \) of structure \( A \); the database \( D_I \) of \( I \) is defined as follows (the first three correspond to \( T \) whereas the other three to \( (\pi_n)_{n \in T} \)):

- leaf(\( n \)) \( \in D_I \) iff \( n \) is a leaf of \( T \);
- root(\( n \)) \( \in D_I \) iff \( n \) is the root of \( T \);
- succ_0(n_1, n_2), succ_1(n_1, n_2) \( \in D_I \) iff \( n_1, n_2 \) is respectively the left, right child of \( n \) in \( T \);
- \( \text{decomp-s}(a_1, \ldots, a_n) \in D_I \) iff \( \overline{\pi}_n = (a_1^1, \ldots, a_n^i) \);
- \( \text{decomp-i}(n, a_i) \in D_I \) iff \( \overline{\pi}_n = (a_1^1, \ldots, a_i^s) \), \( 1 \leq i \leq s \leq w + 1 \);
- \( \text{decomp-I}(n, \{a_i \mid i \in I\}) \in D_I \) iff \( \overline{\pi}_n = (a_1^1, \ldots, a_i^s) \), \( 1 \leq s \leq w + 1 \) and \( I \subseteq \{1, \ldots, s\} \).

**Example 4.1.** Let \( G \) and \( I \) be as in Example 2.1 of page 5. Database \( D_G \) consists of the following facts:

- edge(\( a, b \)), edge(\( a, c \)), edge(\( a, g \)), edge(\( b, c \)), edge(\( c, d \)), edge(\( c, e \)), edge(\( d, e \)), edge(\( f, g \)), \ldots, edge(\( h, g \)).

Database \( D_I \) has the following facts:

- leaf(\( n_3 \)), leaf(\( n_4 \)), leaf(\( n_5 \)); root(\( n_1 \))
- succ_0(\( n_1, n_2 \)), succ_1(\( n_1, n_3 \)), succ_0(\( n_2, n_4 \)), succ_1(\( n_2, n_5 \));
- \( \text{decomp-1}(n_1, a) \), \( \text{decomp-2}(n_1, g) \), \( \text{decomp-3}(n_1, f) \) \ldots
- \( \text{decomp-1}(n_1, \{a\}) \), \( \text{decomp-2}(n_1, \{g\}) \), \( \text{decomp-3}(n_1, \{f\}) \), \( \text{decomp-1,2}(n_1, \{a, g\}) \), \( \text{decomp-1,3}(n_1, \{a, f\}) \), \( \text{decomp-2,3}(n_1, \{g, f\}) \), \( \text{decomp-1,2,3}(n_1, \{a, g, f\}) \) \ldots

The following lemma states that there exists a datalog program that on input database \( D_A \cup D_I \) produces the colors of tree \( T_{I,A} \).

**Lemma 4.1.** Let \( \tau = \{ R_1, \ldots, R_\ell \} \) and let \( w \geq 1 \); there exists a monadic semipositive datalog program \( \Pi_{\tau,w} \) that on input database \( D_A \cup D_I \), corresponding to a \( \tau \)-structure \( A \) having treewidth at most \( w \) and a special tree-decomposition \( I \) of \( A \), produces the colors of \( T_{I,A} = (T, c) \) in the form of facts in the following way

\[
(P_{\tau,w}, D_A \cup D_I) \models \text{color1}-\gamma_1(n), \ldots, \text{color}(\ell + 2)-\gamma_{\ell+2}(n) \quad \text{iff} \quad c(n) = (\gamma_1, \ldots, \gamma_{\ell+2})
\]

The complexity of the program is \((\ell + 2) \cdot |T|\).
Pro. It is not hard to verify that the following program fulfills the desired conditions.

$$\Pi_{r,w} : \begin{align*}
c_{1,s} : & \quad \text{color1-}	ext{s}(x) \leftarrow \text{decomp}^a(x, x_1, \ldots, x_s) \\
c_{2,u,v,P}^k : & \quad \text{color2-}	ext{P}(x) \leftarrow \text{decomp}^w(x, x_1, \ldots, x_u), \text{decomp}^v(y, y_1, \ldots, y_v), \\
& \quad \text{succ}_i(y, x), \{x_i = y_j \mid (i, j) \in P\}, \\
& \quad \{\neg(x_i = y_j) \mid (i, j) \notin P\} \quad \text{for } k = 0, 1 \\
c_2 : & \quad \text{color2-}	ext{root}(x) \\
c_{2+i,s,S_i} : & \quad \text{color}(i + 2)-S_i(x) \leftarrow \text{decomp}^s(x, x_1, \ldots, x_s), \\
& \quad \{r_i(x_{j_1}, \ldots, x_{j_{r_i}}) \mid (j_1, \ldots, j_{r_i}) \in S_i\}, \\
& \quad \{\neg r_i(x_{j_1}, \ldots, x_{j_{r_i}}) \mid (j_1, \ldots, j_{r_i}) \notin S_i\} \quad \text{for } 1 \leq i \leq \ell
\end{align*}$$

where $s, u, v \in W$, $P \in \mathcal{P}(W_u \times W_v)$, $S_1 \in \mathcal{P}(W_w^{*1})$, $\ldots$, $S_\ell \in \mathcal{P}(W_w^{*\ell})$, for $W = \{1, \ldots, w + 1\}$ and $W_i = \{1, \ldots, i\}$.

Complexity. Program $\Pi_{r,w}$ has exactly $(\ell + 2) \cdot |T|$ instantiations. \qed

4.3 Evaluation of unary MSO-definable queries via monadic datalog

MSO unary queries over structures of bounded treewidth are the queries defined by MSO-r formulas $\phi(x)$. By Theorem 3.1, there exists a decomposition automaton $w$-dec-$\phi_A$ such that for every $\tau$-structure $A$ having treewidth at most $w$ and special tree-decomposition $T$ of $A$ we have

$$a \in \phi(A) \iff a \in w$-$\text{dec-}\phi_A(T_{\tau,A})$$

That is, $a \in \phi(A)$ iff there exists a successful run of $w$-dec-$\phi_A$ over $T_{\tau,A}$ having as assignment part $\varepsilon_a : T \rightarrow \mathcal{P}(W)$ Recall that $\varepsilon_a$ is defined as follows: $\varepsilon_a(n) = \emptyset$ for all $n \neq \text{node}_T(a)$, and $\varepsilon_a(n) = \{i\}$ iff $\text{pair}_T(a) = (n, i)$. Notice that all assignment parts of successful runs of $w$-dec-$\phi_A$ have this form. Thus, for every $a \in A$ with $\text{pair}_T(a) = (n, i)$ we have that $a \in \phi(A)$ iff there exists a successful run of $w$-dec-$\phi_A$ over $T_{\tau,A}$ assigning an $n$ state $q\{i\}$, for some $q \in Q$. By definition, if $\text{pair}_T(a) = (n, i)$ then $a = a^n_i$. 5 Thus we conclude that: $a^n_i$ is a satisfying assignment of $\phi(x)$ on $A$ iff there exists a successful state of $w$-dec-$\phi_A$ over $T_{\tau,A}$ of the form $q\{i\}$ at node $n$. Therefore, as in the case of trees, it suffices to have datalog rules that compute successful states and collect elements $a^n_i$ for each successful state of the form $q\{i\}$ at node $n$. This leads to an optimized datalog program for unary queries which is monadic.

Recall that in the case of trees and unary queries (see Section 4.3) we had the following: \{n\} is a satisfying assignment of $\phi$ where iff node $n$ has a successful state of the form $(q, 1)$. So, it was also sufficient there to compute successful states. The programs $(P' \cup S')$ in that case, and $(P^* \cup S^*)$ here) producing successful states of tree-automata are the same in both cases.

5We remind that $a^n_i$ is the $i$-th element of the block $\tau_i$ corresponding to the node $n$ of the given special tree-decomposition.
Proposition 4.1. Let $\tau = \{R_1, \ldots, R_n\}$; for every MSO[$\tau$] formula $\phi(x)$ and integer $w \geq 1$, we can construct a monadic semipositive datalog query $Q^w_\phi = (\Pi^w_\phi, \text{assign})$ such that for every $\tau$-structure $A$ having treewidth at most $w$ and special tree-decomposition $T = (T, (\pi_n)_{n \in T}$, \[ a \in \phi(A) \iff a \in Q^w_\phi(D_A \cup D_T) \]
The complexity of the evaluation is at most $(\ell + 2 + 2^{w+2} \cdot s^2) \cdot |T| + a$ where $s$ is the size of set $Q$ of $\phi$-states, $m$ is the size of $T$ and $a = |\phi(T)|$.

Proof. $\Pi^w_\phi = \Pi_{r,w} \cup P^* \cup S^* \cup_{i,q} r_{q,i}$: Program $\Pi_{r,w}$ produces the coloring of $T_{I,A}$ which it is then used as input database (together with the tree structure $T$ of $T_{I,A}$) to program $P^* \cup S^*$ to compute the successful states of $w$-dec-$\text{Assign}_\phi$ over $T_{I,A}$; finally, rules $r_{q,i}$ decode these successful states (with the help of $D_T$) to elements of $A$ which constitute $\phi(A)$.

More precisely, program $\Pi_{r,w}$ (defined in Lemma 4.1) produces the colors of the tree $T_{I,A}$ encoding $\tau$-structure $A$ and special tree-decomposition $T$: consider that facts $\text{color}1$-$\gamma_1(n), \ldots, \text{color}(\ell+2)$-$\gamma_{\ell+2}(n)$ are produced by $\Pi_{r,w}$ iff $\gamma = (\gamma_1, \ldots, \gamma_{\ell+2})$ is the color of $n$ in $T_{I,A}$; this is the only part of $\Pi^w_\phi$ where negation occurs. Program $P^* \cup S^*$ performs a bottom-up traversal of tree $T_{I,A}$ producing the potential states of $w$-dec-$\text{Assign}_\phi$ at each node in $T$. Then, a top-down traversal is performed by $S^*$ which produces -using potential states- successful states: a fact $\text{succ}-qI(n)$ is derived from $S^*$ if there exists a successful run of $w$-dec-$\text{Assign}_\phi$ over $T_{I,A}$ assigning state $qI$ at node $n$. As it has already been stated, for each successful state $qI$ we either have $I = \emptyset$ or $I = \{i\}$ for some $i$; more precisely, if a state of the form $q(I)$ is assigned at node $n$ via a successful run of $w$-dec-$\text{Assign}_\phi$ over $T_{I,A}$, then $a^i_n$ is a satisfying assignment; with rules $r_{q,i}$ we distinguish such successful states and convert them to the corresponding satisfying assignment $a^i_n$.

\[ P^* \quad p^*_qI : \quad \text{pot}-qI(x) \leftarrow \text{leaf}(x), \{\text{color}j-\gamma_j(x) \mid 1 \leq j \leq \ell + 2\} \]
\[ p^*_qI_1 : \quad \text{pot}-qI(x) \leftarrow \text{pot}-qI_1(x_1), \text{pot}-qI_2(x_2), \text{succ}0(x, x_1), \text{succ}1(x, x_2), \]
\[ \{\text{color}j-\gamma_j(x) \mid 1 \leq j \leq \ell + 2\} \]
\[ S^* \quad s^*_qI : \quad \text{succ}-qI(x) \leftarrow \text{pot}-qI(x), \text{root}(x) \]
\[ s^*_qI_1,j : \quad \text{succ}-qI_1(x_j) \leftarrow \text{succ}-qI(x), \text{pot}-qI_1(x_1), \text{pot}-qI_2(x_2), \text{succ}0(x, x_1), \text{succ}1(x, x_2), \]
\[ \{\text{color}j-\gamma_j(x) \mid 1 \leq j \leq \ell + 2\} \quad \text{for} \quad j = 1, 2 \]
\[ r_{q,i} : \quad \text{assign}(x_i) \leftarrow \text{succ}-qI(i)(x), \text{decomp}-i(x, x_i) \]

where $q \in F$ in rules $s^*_qI$ and $q \in Q$ in rules $r_{q,i}$; $qI \in w-D_0, \gamma qqI_2 I \in w-D, \gamma = (\gamma_1, \ldots, \gamma_{\ell+2})$; $I, I_1, I_2 \subseteq W$ and $i \in W$.

Complexity. Program $\Pi_{r,w}$ has exactly $(\ell+2) \cdot |T|$ instantiations. Then, during the evaluation of $P^*$, we have for each leaf exactly $2^{w+1}$ instantiations (as many as the number of different sets $I \in \mathcal{P}(W)$) and for each inner node at most $2^{w+1} \cdot s^2$ instantiations (i.e. as many as $|\mathcal{P}(W)| \times |Q|^2$); analogous is the number of instantiations of $S^*$. Finally, there are $a = |\phi(A)|$ instantiations of rules of the family $r_{q,i}$; thus, the complexity of the evaluation of the program is at most $(\ell + 2 + 2^{w+2} \cdot s^2) \cdot |T| + a$.

Immediate consequence of Proposition 4.1 is the following definability result.

Theorem 4.1. The class of unary MSO-definable queries over structures of bounded treewidth is monadic-datalog definable.
4.4 Evaluation of \( k \)-ary MSO formulas via datalog of maximum arity \( k + 1 \)

As we have already mentioned, the datalog solutions that we propose in this work are based on the direct reduction of the initial MSO evaluation problem over structures of bounded treewidth to a tree-automata evaluation problem defined via the assignment (decomposition) automata notion. Indeed, datalog programs presented in this section implement the following procedure: they gradually compose, traversing the input tree in a bottom-up manner, successful runs of \( w \)-decomposition assignment automata, and at the same time they perform a decoding of the assignment vectors of these runs; \( \phi(A) \) is comprised of these decoded assignment vectors which are finally collected in predicates \( \text{assign} \).

More precisely, let \( \phi \) be a \( k \)-ary MSO[\( \tau \)] formula that we want to evaluate over a \( \tau \)-structure \( A \); by Theorem 3.1, there exists a decomposition automaton \( w\text{-dec-Assign}_\phi \) such that for every \( \tau \)-structure \( A \) and special tree-decomposition \( T \) of \( A \) with width \( w \) we have \( \phi(A) = w\text{-dec-Assign}_\phi(T_{I_{\phi}}A) \). Let \( \text{a-suc-runs}_\phi(T_{I_{\phi}}A) \) denote the set of assignment parts of successful runs of \( w\text{-dec-Assign}_\phi \) over \( T_{I_{\phi}}A \), i.e. \( \text{a-suc-run}_\phi(T_{I_{\phi}}A) = \{ \varepsilon \mid \text{there exists a suc. run } \rho \text{ of } w\text{-dec-Assign}_\phi \text{over } T_{I_{\phi}}A \text{ s.t. } \rho = \varsigma; \varepsilon \} \); and let \( \varepsilon_i \subseteq W \) denote the \( i \)-th element of \( \varepsilon(n) = (\varepsilon_{i1}, \ldots, \varepsilon_{ik}) \); we also remind that \( a_i \) denotes the \( j \)-th element of the bag \( T_n \) of the tree decomposition \( T \) and that \( T \) is the set of nodes of \( T_{I_{\phi}}A \). Then, by Definitions 3.2 & 3.3 and previously mentioned equality, the following holds: if \( B_{\varepsilon,n} = \{ a_i^j \mid j \in \varepsilon_i^k \} \) for \( 1 \leq i \leq k \) (notice that each \( B_{\varepsilon,n}^j \) corresponds to the elements of the projection \( \pi_{\varepsilon,n} \)), then

\[
\phi(A) = \{ \bigcup_{n \in T} (B_{\varepsilon,n}^1, \ldots, B_{\varepsilon,n}^k) \mid \varepsilon \in \text{a-suc-runs}_\phi(T_{I_{\phi}}A) \}
\]

where the union over \( n \)'s is component-wise. Via the above equation, we obtain the following straightforward automata algorithm for the computation of \( \phi(A) \):

- Compute, traversing in a bottom-up manner tree \( T_{I_{\phi}}A \), the assignments encoded in the assignment parts of all possible runs of \( w\text{-dec-Assign}_\phi \) over \( T_{I_{\phi}}A \) (we shall denote this set of assignment parts as \( a\text{-runs}_\phi(T_{I_{\phi}}A) \)); more precisely, for all such \( \varepsilon \)'s, compute at each node \( n \), sets \( \overline{B}_{\varepsilon,T_n} = \bigcup_{n \in T} \overline{B}_{\varepsilon,n} \) where \( \overline{B}_{\varepsilon,n} = (B_{\varepsilon,n}^1, \ldots, B_{\varepsilon,n}^k) \), via the following component-wise unions (supposing that node \( n \) has children \( n_1, n_2 \))\(^6\)

\[
\overline{B}_{\varepsilon,T_n} = \overline{B}_{\varepsilon,n} \cup \overline{B}_{\varepsilon,T_{n_1}} \cup \overline{B}_{\varepsilon,T_{n_2}} \quad (*)
\]

where \( T_n \subseteq T \) is the domain of the subtree of \( T_{I_{\phi}}A \) rooted at \( n \). Let \( B_n \subseteq A \) be the union of bags \( a_n \) for all \( n \in T_n \) i.e. \( B_n = \bigcup_{n \in T_n} a_n \) ; computed set \( \overline{B}_{\varepsilon,T_n} \) corresponds to the restriction of assignment \( \overline{B}_{\varepsilon} \) on \( B_n \), that is \( \overline{B}_{\varepsilon,T_n} = \overline{B}_{\varepsilon} \cap (B_n)^k \) where the intersection is component-wise.

- Select only the assignments that correspond to assignments parts of runs that are successful; we distinguish the ones that correspond to successful runs from the ones that correspond to non-successful runs via the value of the state part at the root. Notice that this is the only value of the state part \( \varsigma \) of a run \( \varsigma; \varepsilon \) that we need in order to tell whether \( \overline{B}_{\varepsilon} \) is a satisfying assignment or not: \( \overline{B}_{\varepsilon} \) is a satisfying assignment iff \( \varsigma(r) \in F \).

In terms of datalog the first of the above two steps is implemented via rules \( r_{\gamma q} \) and \( r_{\gamma q q} \) of program \( A \) below. More precisely, fact \( \text{assign-}q(n, \overline{B}) \) is computed iff there exists a run \( \varsigma; \varepsilon \)

\(^6\)Obviously when \( n \) is a leaf \( \overline{B}_{\varepsilon,T_n} = \overline{B}_{\varepsilon,n} \)
of $w$-dec-$\text{Assign}_\phi$ over $\mathcal{T}_{\mathcal{L},A}$ s.t. $\zeta(n) = q$ and $\overline{B}_\varepsilon|_{B_n} = \overline{B}$. Obviously rules $s_q$ of below given datalog program $A$ implement the second step of the above described algorithm.

$$
A.\quad r_{\gamma q T} : \quad \text{assign-}q(x, \overline{x}) \leftarrow \text{"color-}\gamma(x)\text{"},\text{leaf}(x), \text{"decomp-}\mathcal{I}(x, \overline{x})"
$$

$$
r_{\gamma q q_1 q_2 T} : \quad \text{assign-}q(x, \overline{x}) \leftarrow \text{"color-}\gamma(x)\text{"},\text{succ}_0(x, u_1), \text{assign-}q_1(u_1, \overline{y}), \text{succ}_1(x, u_2),\text{assign-}q_2(u_2, \overline{x}), \text{decomp-}\mathcal{I}(x, \overline{x}), \text{cw-union}(\overline{x}, \overline{y}, \overline{z})
$$

$$s_q : \quad \text{assign}(\overline{x}) \leftarrow \text{assign-}q(x, \overline{x}), \text{root}(x)$$

where $\gamma q T \in w$-$D_0$, $\gamma q q_1 q_2 T \in w$-$D$, $q \in F$ in the last rule; $\overline{x} = x_1, \ldots, x_k$, $\overline{y} = y_1, \ldots, y_k$, $\overline{z} = z_1, \ldots, z_k$, $\mathcal{I} = (I_1, \ldots, I_k)$; where “color-}$\gamma(x)$“ abbreviates the following sequence of facts $\text{color}1$-$\gamma_1(x), \ldots, \text{color} \ell + 2$-$\gamma_{\ell+2}(x)$; “decomp-$\mathcal{I}(x, \overline{x})$“ abbreviates $\text{decomp-}\mathcal{I}_1(x, x_1), \ldots, \text{decomp-}\mathcal{I}_k(x, x_k)$.

Optimized datalog solution. In fact, the datalog program that we shall use to evaluate our MSO queries and that is given in the following proposition, constitutes an optimized version of the above given datalog program $A$ being the result of resolution-based filtering rewriting (also known as “magic-sets" method) where the successful states are used properly as filters in body of the rules of $A$. Successful states of $w$-dec-$\text{Assign}_\phi$ over $\mathcal{T}_{\mathcal{L},A}$ filter out all non-satisfying assignments from the beginning; recall that $A$ was able to do so only at the root.

More precisely, filtered stratum $A^*_S$ of program $\Pi^w_{\phi,k}$ computes (in its goal predicate assign) the satisfying assignments of $\phi$ performing component-wise unions of $k$-tuples over set $A$ (implementing equation (*) in a bottom-up manner. More precisely, fact assign-$q(n, \overline{B})$ is computed by $A^*_S$ iff there exists a successful run $\zeta; \varepsilon$ of $w$-dec-$\text{Assign}_\phi$ over $\mathcal{T}_{\mathcal{L},A}$ s.t. $\zeta(n) = q$ and $\overline{B}_\varepsilon|_{A_n} = \overline{B}$, where $A_n = \{ a \in A \mid \text{node}(a) \in T_n \}$ (i.e. computed tuples $\overline{B}$ constitute restrictions of satisfying assignments of $\phi$ on set $A_n \subseteq A$).

Notice that we need to extend our input database with predicates corresponding to union operation; more precisely we use an extended database over the powerset of the domain $A$ of input structure $\mathcal{A}$: this database, denoted $U_d$, contains the following facts that define unions between subsets of $A$: $\text{union}(A \cup A_1 \cup A_2, A, A_1, A_2)$, for $A \subseteq a_n$ and $A_1 \subseteq B_{n_1}, A_2 \subseteq B_{n_2}, n \in T$. We define $D^U_{A,\mathcal{I}} = D_d \cup D_\mathcal{I} \cup U_d$.

Proposition 4.2. Let $\tau = \{ R_1, \ldots, R_\ell \}$; for every MSO$[\tau]$ formula $\phi(X_1, \ldots, X_k)$ and positive integer $w$, we can construct a semipositive datalog query $Q^w_{\phi,k} = (\Pi^w_{\phi,k}, \text{assign})$ of arity $k + 1$ such that for every $\tau$-structure $\mathcal{A}$ having treewidth at most $w$, and special tree-decomposition $\mathcal{T} = (\mathcal{T}, (\tau_n)_{n \in T})$ of $\mathcal{A}$,

$$(B_1, \ldots, B_k) \in \phi(\mathcal{A}) \iff (B_1, \ldots, B_k) \in Q^w_{\phi,k}(D^U_{A,\mathcal{I}})$$

The complexity of the evaluation is at most $(\ell + 2 + 2^k\cdot(w+1)+1 \cdot s^2 + a) \cdot |\mathcal{T}| + a$ where $s$ is the size of set $Q$ of $\phi$-states, $m$ is the size of $\mathcal{T}$ and $a = |\phi(\mathcal{T})|$. 

23
Proof. $\Pi_{w,k}^w = \Pi_{r,w} \cup P \cup S \cup A_S^*$: Program $\Pi_{r,w}$ produces the coloring of $T_{I,A}$ which is in turn used as input database (together with the tree structure of $T_{I,A}$) to program $P^* \cup S^*$ to compute the successful states of $w$-dec-Assign over $T_{I,A}$; successful states filter the computation of assignments in the rules of $A_S^*$ in the sense of the analysis given above.

In the rules defined below, $γ = (γ_1, \ldots, γ_{ℓ+2})$ and $T = (I_1 \ldots I_k) \in (P(W))^k$.

\[ P. \quad p_{γq} : \quad \text{pot-}q(x) ← \text{leaf}(x), \{\text{colori-}γ_i(x) \mid 1 ≤ i ≤ ℓ + 2\} \]

\[ p_{γqq1q2} : \quad \text{pot-}q(x) ← \text{pot-}q_1(x_1), \text{pot-}q_2(x_2), \text{succ}_0(x, x_1), \text{succ}_1(x, x_2), \]

\[ \{\text{colori-}γ_i(x) \mid 1 ≤ i ≤ ℓ + 2\} \]

\[ S. \quad s_q : \quad \text{succ-}q(x) ← \text{pot-}q(x), \text{root}(x) \]

\[ s_{γqq1q2}^i : \quad \text{succ-}q_i(x_i) ← \text{succ-}q(x), \text{pot-}q_1(x_1), \text{pot-}q_2(x_2), \text{succ}_0(x, x_1), \text{succ}_1(x, x_2), \]

\[ \{\text{colori-}γ_i(x) \mid 1 ≤ i ≤ ℓ + 2\} \quad i = 1, 2 \]

$q ∈ F$ in $s_q$, $γq ∈ w-D'_0$ and $γqq1q2 ∈ w-D'$.

\[ A_S^*. \quad g_{γqT} : \quad \text{assign-}q(x, x_1, \ldots, x_k) ← \text{succ-}q(x), \{\text{colori-}γ_i(x) \mid 1 ≤ i ≤ ℓ + 2\}, \text{leaf}(x), \]

\[ \text{decomp-I}_1(x, x_1), \ldots, \text{decomp-I}_k(x, x_k) \]

\[ g_{γqq1q2T} : \quad \text{assign-}q(x, x_1, \ldots, x_k) ← \text{succ-}q(x), \{\text{colori-}γ_i(x) \mid 1 ≤ i ≤ ℓ + 2\}, \]

\[ \text{succ}_0(x, u_1), \text{assign-}q_1(u_1, y_1, \ldots, y_k), \text{succ}_1(x, u_2), \text{assign-}q_2(u_2, z_1, \ldots, z_k), \]

\[ \text{decomp-I}_1(x, v_1), \ldots, \text{decomp-I}_k(x, v_k) \]

\[ \text{union}(x_1, v_1, y_1, z_1), \ldots, \text{union}(x_k, v_k, y_k, z_k) \]

\[ a_q : \quad \text{assign}(x_1, \ldots, x_k) ← \text{assign-}q(x, x_1, \ldots, x_k), \text{root}(x) \]

where $γqT ∈ w-D_0$, $γqq1q2T ∈ w-D$, $q ∈ Q$.

Complexity. Program $\Pi_{r,w}$ has exactly $(ℓ + 2) \cdot |T|$ instantiations; Then, during the evaluation of each of $P$ and $S$, we have at each node at most $2^{κ_{(w+1)} \cdot s^2}$ instantiations (i.e. as many as $|(P(W))^k| \times |Q|^2$). Finally, we have $|φ(A)|$ instantiations of rules at each node when we evaluate $A_S^*$ and another $|φ(A)|$ at the root; thus, the complexity of the evaluation of the program is at most $(ℓ + 2 + 2^{κ_{(w+1)} \cdot s^2 + a}) \cdot |T| + a$. \hfill \Box

Immediate consequence of Proposition 4.2 is the following definability result generalizing the one given in [14] for the case of trees.

**Theorem 4.2.** The class of $k$-ary MSO-definable queries over relational structures of bounded treewidth is $(k + 1)$-datalog definable.

5 MSO-definable optimization problems & their datalog solutions

The datalog programs given in this section are in fact variants of the programs of the previous section solving the MSO evaluation problem.
a. Evaluating formula \( \psi^{\exists k} = \exists^k X \phi(X) \) which is true if there exists a set \( X \) of size \( k \) satisfying MSO formula \( \phi(X) \).

**Proposition 5.1.** Let \( \tau = \{ R_1, \ldots, R_\ell \} \); for every formula \( \psi^{\exists k} = \exists^k X \phi(X) \) and positive integer \( w \), we can construct a monadic datalog\(^*\) query \( Q^{\exists k,w}_\psi = (\Pi^{\exists k,w}_\psi, \psi\text{-true}) \) such that for every \( \tau \)-structure \( A \) of size at most \( m \) and with special tree-decomposition \( \mathcal{I} = (\mathcal{T},(\mathcal{I}_n))_{n \in \mathcal{T}} \) of width \( w \),

\[
A \models \psi^{\exists k} \text{ iff } \Pi^{\exists k,w}_\psi(D_A \cup D_{\mathcal{I}}) \models \psi\text{-true}.
\]

The complexity of the evaluation is at most \((\ell + 2 + 2^{w+1} \cdot s^2) \cdot |\mathcal{T}|\).

**Proof.** \( \Pi^{\exists k,w}_{\phi,m} = \Pi_{\tau,w} \cup P^{c,w}_m \cup A^k \): program \( \Pi_{\tau,w} \) (defined in Lemma 4.1) produces the colors of the tree \( \mathcal{T}_{\tau,A} \) encoding \( \tau \)-structure \( A \) and special tree-decomposition \( \mathcal{I} \); datalog program \( P^{c,w}_m \) defined below produces fact \( \text{pot-qi}(n) \) iff there exists a run \( \zeta;\epsilon \) of \( w\text{-dec}\text{-Assign}_\phi \) over \( \mathcal{T}_{\tau,A} \) such that \( \zeta(n) = q \) and \( i = \Sigma_{n \in \mathcal{T}_n} |\epsilon(n)| \). Note that if \( \zeta;\epsilon \) is a successful run, then \( q \) is a successful state and \( i \) is the size of the restriction on \( A_n \) of the satisfying assignment \( \beta_\epsilon \). Thus, the production at the root of a pair \((q,k)\) with \( q \) being a final state, proves the existence of a satisfying assignment for \( \phi(X) \) that has size \( k \) i.e. that \( \psi^{\exists k} \) is true. This latter requirement is checked via rules of \( A^k \) below.

\[
\begin{align*}
P^{c,w}_m & \quad \text{cp}_{\gamma q I} : \text{pot-qi}(x) \leftarrow \text{leaf}(x), \{\text{color}-\gamma_j(x) \mid 1 \leq j \leq \ell + 2\}, \: |I| = i \\
\text{cp}_{\gamma q q_1 q_2 I, i_1 i_2} & : \text{pot-qi}(x) \leftarrow \text{pot-qi}(x_1), \text{pot-qk}_2(x_2), \text{success}(x_1), \text{success}(x_2), \\
& \quad \{\text{color}-\gamma_j(x) \mid 1 \leq j \leq \ell + 2\}, \: i = |I| + i_1 + i_2 \\
A^k & \quad \text{a}_{q k} : \text{\psi\text{-true}} \leftarrow \text{pot-qk}(x), \text{root}(x).
\end{align*}
\]

where rules \( \text{cp}_{\gamma q I} \) are defined for all \( \gamma q I \in w\text{-D}_0 \); rules \( \text{cp}_{\gamma q q_1 q_2 I, i_1 i_2} \) are written for \( \gamma q q_1 q_2 I, i_1 i_2 \) s.t. \( \gamma q q_1 q_2 I \in w\text{-D} \) and \( i_1, i_2 \) positive integers such that \( i = |I| + i_1 + i_2 \leq m \); rules \( \text{a}_{q k} \) are written for all \( q \in F \). \( \square \)

b. Evaluating formula \( \text{"card-}^{\text{min}\text{/max}}{^\phi} \) whose (unique) satisfying assignment is the cardinality of satisfying assignments of \( \phi(X) \) having minimum (resp. maximum) size.

**Proposition 5.2.** Let \( \tau = \{ R_1, \ldots, R_\ell \} \); for every formula \( \text{card-}^{\text{min}\text{/max}}{^\phi} \) and positive integer \( w \), we can construct a monadic stratified datalog\(^*\) program \( \Pi^{w,m}_{\text{card min } \phi} \) such that for every \( \tau \)-structure \( A \) of size at most \( m \) and with special tree-decomposition \( \mathcal{I} = (\mathcal{T},(\mathcal{I}_n))_{n \in \mathcal{T}} \) of width \( w \),

\[
k \text{ satisfies card-}^{\text{min}\text{/max}}{^\phi} \text{ over } A \text{ iff } \Pi^{w,m}_{\text{card min } \phi}(D_A \cup D_{\mathcal{I}}) \models \text{min-}k.
\]

The complexity of the evaluation is at most \((\ell + 2 + 2^{w+1} \cdot s^2 + 2 \cdot s) \cdot |\mathcal{T}|\).

**Proof.** We define \( \Pi^{w,m}_{\text{card min } \phi} \) by the union \( \Pi_{\tau,w} \cup P^{c,w}_m \cup A^{c,\text{min}}_m \) where \( \Pi_{\tau,w} \) and \( P^{c,w}_m \) are defined in Lemma 4.1 and Proposition 5.1 respectively; \( A^{c,\text{min}}_m \) is defined as follows:

\[
A^{c,\text{min}}_m, \quad s_{q k} : \quad \text{suc-}k \leftarrow \text{pot-qk}(x), \text{root}(x)
\]

25
where \( q \in F \) and \( k \leq m \).

After the bottom-up evaluation of potential states via program \( P_{m}^{w} \), rules \( s_{qk} \) of \( A_{m}^{\min} \) collect in predicates \( \text{suc-}k \) the sizes \( k \) of sets being the satisfying assignments of \( \phi \) over the input structure. Finally, rules \( \text{min}_{k} \) compute the minimum of the set containing the sizes of the satisfying assignments of \( \phi \), providing thus the solution to the considered evaluation problem.\( \square \)

c. Evaluating formula \( "\phi^{\min/\max}(X)^{n} \) whose satisfying assignments are the satisfying assignments of \( \phi(X) \) having minimum (resp. maximum) size.

**Proposition 5.3.** Let \( \tau = \{R_{1}, \ldots, R_{n}\} \); for every formula \( \phi^{\min}(X) \) and positive integer \( w \), we can construct a stratified datalog \( \gamma \) program \( \Pi^{w,m}_{\phi} \) of arity 2 such that for every \( \tau \)-structure \( A \) of size at most \( m \) and with special tree-decomposition \( \mathcal{I} = (T, (\pi_{n})_{n \in T}) \) of width \( w \),

\[
B \text{ satisfies } \phi^{\min}(X) \text{ over } A \quad \text{iff} \quad \Pi^{w,m}_{\phi}(\mathcal{D}_{A} \cup \mathcal{D}_{T}) \models \text{assign}(B).
\]

The complexity of the evaluation is at most \((\ell + 2 + 2^{w+2} \cdot s^2 + 2 \cdot s + a) \cdot |T| + a\).

**Proof.** We define \( \Pi^{w,m}_{\phi} \) by the union \( \Pi^{w,m}_{\phi} \cup S_{m}^{c,w} \cup A_{S_{m}^{c,w}} \) where \( \Pi^{w,m}_{\phi} \) is defined in Proposition 5.2; \( S_{m}^{c,w} \) and \( A_{S_{m}^{c,w}} \) are defined as follows:

\[
S_{m}^{c,w}. \quad cs_{qk} : \quad \text{suc-}qk(x) \leftarrow \text{min-}k, \text{pot-}qk(x), \text{root}(x)
\]

\[
\begin{align*}
\text{cs}_{qj}^{\gamma q_{1}q_{2}I,i_{1}i_{2}} : & \quad \text{suc-}q_{i}j_{i}(x_{j}) \leftarrow \text{suc-}q_{i}j_{i}(x), \text{pot-}q_{1i_{1}}(x_{1}), \text{pot-}q_{2i_{2}}(x_{2}), \text{succ}(x, x_{1}), \text{succ}(x, x_{2}), \\
& \{\text{color-}r_{j}(x) \mid 1 \leq j \leq \ell + 2\} \text{ for } j = 1, 2 \quad i = |I| + i_{1} + i_{2}
\end{align*}
\]

\[
A_{S_{m}^{c,w}}. \quad g_{q_{1}} : \quad \text{assign-}q_{i}(x, y) \leftrightarrow \text{suc-}q_{i}(x), \{\text{color-}r_{i}(x) \mid 1 \leq i \leq \ell + 2\}, \text{leaf}(x), \\
& \text{decomp-}I(x, y) \quad |I| = i
\]

\[
\begin{align*}
g^{\gamma q_{1}q_{2}I,i_{1}i_{2}} : & \quad \text{assign-}q_{i}(x, y) \leftrightarrow \text{suc-}q_{i}(x), \{\text{color-}r_{i}(x) \mid 1 \leq i \leq \ell + 2\}, \\
& \text{succ}(x, x_{1}), \text{assign-}q_{1i_{1}}(x_{1}, y_{1}), \\
& \text{succ}(x, x_{2}), \text{assign-}q_{2i_{2}}(x_{2}, y_{2}), \\
& \text{decomp-}I(x, y), \text{union}(y, v, y_{1}, y_{2}) \quad i = |I| + i_{1} + i_{2}
\end{align*}
\]

\[
a_{qk} : \quad \text{assign}(y) \leftarrow \text{assign-}qk(x), \text{root}(x)
\]

where rules \( cs_{qk} \) and \( a_{qk} \) are written for all \( q \in F \) and \( k \leq m \); rules \( cs^{\gamma q_{1}q_{2}I,i_{1}i_{2}} \) and \( g^{\gamma q_{1}q_{2}I,i_{1}i_{2}} \) are written for \( \gamma q_{1}q_{2}I,i_{1}i_{2} \) s.t. \( \gamma q_{1}q_{2}I \in w-D \) and \( i_{1}, i_{2} \) positive integers such that \( i = |I| + i_{1} + i_{2} \leq m \). \( \square \)

6 A case study: vertex cover

A **vertex cover** of a graph \( G \) is a set \( A \) of vertices such that each edge of \( G \) is incident to at least one vertex in \( A \). A minimum vertex cover is a vertex cover of smallest possible size.
The problem of finding a minimum vertex cover is a classical NP-hard optimization problem in computer science; its decision version, the vertex cover problem, is also a classical NP-complete problem in computational complexity theory.

In this section we apply the automata and datalog approaches presented in the previous sections for the case of MSO[τ\_G] formula

\[ \phi(Z) = \forall x \forall y (E(x, y) \rightarrow \text{In}(x, Z) \lor \text{In}(y, Z)) \]

expressing that set \( Z \) is a vertex cover of a graph \( G \).

A decomposition-automaton for vertex cover. We apply the constructions given in the proof of main automata theorem (Theorem 3.1) for the following equivalent form of vertex cover formula

\[ \phi(Z) \equiv \neg \exists x \exists y (E(x, y) \land \neg \text{In}(x, Z) \land \neg \text{In}(y, Z)) \]

The vertex-cover automaton is given in Table 7. Note that conditions \( \text{vertex}(i), \text{AdgInPar}(i, j), \text{Edge}(i, j) \) used in the corresponding transitions definitions correspond exactly to conditions \( \text{element}(i), \text{RInPar}(i, j), \text{R}(i, j) \) defined in Table 1 but have been renamed to agree with their meaning in the case of graphs; the updated definition are given in Table 5.

<table>
<thead>
<tr>
<th>Conditions/Sets</th>
<th>w.r.t. ( \gamma \in \Gamma_{\tau,w} )</th>
<th>w.r.t. ( \mathcal{I}, \mathcal{G} ) at ( n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{vertex}(j) )</td>
<td>iff ( j \leq \gamma_1 \land \forall i \in W ((j, i) \not\in \gamma_2) )</td>
<td>iff ( j \in \text{Pos}_\mathcal{I}(n) )</td>
</tr>
<tr>
<td>( V_\gamma )</td>
<td>:= ( {i \mid \text{vertex}(i)} )</td>
<td>:= ( \text{Pos}_\mathcal{I}(n) )</td>
</tr>
<tr>
<td>( \text{set}(I) )</td>
<td>iff ( I \subseteq V_\gamma )</td>
<td>iff ( I \subseteq \text{Pos}_\mathcal{I}(n) )</td>
</tr>
<tr>
<td>( \text{SameInPar}(J) )</td>
<td>:= ( {i \mid \exists j \in J ((j, i) \in \gamma_2)} )</td>
<td>:= ( {i \mid \exists j \in J (a_i^j = a_p^j)} )</td>
</tr>
<tr>
<td>( \text{Adj}(j) )</td>
<td>iff ( I = {i \mid \exists j \in J ((j, i) \in \gamma_2)} )</td>
<td>if ( \text{Adj}(j) ) := ( {i \mid (j, i) \in \gamma_3} )</td>
</tr>
<tr>
<td>( \text{AdjInPar}(j) )</td>
<td>:= ( \text{SameInPar}(R(j)) )</td>
<td></td>
</tr>
<tr>
<td>( \text{Edge}(j, \ell) )</td>
<td>iff ( \text{vertex}(j) \land \text{vertex}(\ell) \land (j, \ell) \in \gamma_3 )</td>
<td>iff ( j \in \text{Pos}<em>\mathcal{I}(n) \land \ell \in \text{Pos}</em>\mathcal{I}(n) \land (a_i^j, a_p^\ell) \in E^\mathcal{G} )</td>
</tr>
<tr>
<td>( \text{NotEdge}_{\gamma \setminus J} )</td>
<td>:= ( \bigcup_{j \in V_\gamma \setminus J} \text{AdjInPar}(j) )</td>
<td></td>
</tr>
<tr>
<td>( \text{Adj}_{\gamma \setminus J} )</td>
<td>iff ( \exists j, \ell \notin J(\text{Edge}(j, \ell)) )</td>
<td></td>
</tr>
<tr>
<td>( \text{NotEdge}_{\gamma \setminus J} )</td>
<td>iff ( \forall j, \ell \in V_\gamma \setminus J \neg (\text{Edge}(j, \ell)) )</td>
<td></td>
</tr>
</tbody>
</table>

Example 6.1. We shall give the successful runs of the “counting” automaton for vertex-cover w.r.t. to graph and tree decomposition of 1; notice that we write the states in the form \((I, q, i)\).

Before that, we give some of the conditions satisfied at each node:
Table 6: The meaning of states of $w$-dec-$\text{Assign}_\phi(V_C(Z))$.

<table>
<thead>
<tr>
<th>State</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Q_f$</td>
<td>failure of acceptance: a non satisfying assignment has been selected</td>
</tr>
<tr>
<td>$Q^I$</td>
<td>there exists a set $A \subseteq V_n \setminus Z_n$, i.e. of vertices in $V_n$ that have not been selected via the current run over $T_n$, and set ${a^i_p \mid i \in I}$ consists of the adjacent vertices of vertices of $A$ that occur in $V \setminus V_n$</td>
</tr>
</tbody>
</table>

Let $V_n = \{a \in V \mid \text{node}(a) \in T_n\}$ and let $Z_n \subseteq V_n$ be the set vertices that have been selected at $T_n$ via the automaton. We consider that states are assigned at $n$; $p$ is the parent of $n$.

Table 7: Deterministic automaton for vertex cover formula $\phi_{VC}(Z)$.

<table>
<thead>
<tr>
<th>Transition</th>
<th>Conditions satisfied at current node $n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\gamma, L \mapsto Q^I$</td>
<td>$\text{set}(L) &amp; \text{NotEdge}<em>{\gamma \setminus L} &amp; \text{AdjInPar}</em>{\gamma \setminus L} = I$</td>
</tr>
<tr>
<td>$Q^I, Q^S, \gamma, L \mapsto Q^I$</td>
<td>$\text{set}(L) &amp; \text{NotEdge}<em>{\gamma \setminus L} &amp; (J \cup S) \cap (V</em>\gamma \setminus L) = \emptyset &amp; \text{SameInPar}(J \cup S) \cup \text{AdjInPar}_{\gamma \setminus L} = I$</td>
</tr>
<tr>
<td>$q_1, q_2, \gamma, I \mapsto Q_f$</td>
<td>all cases that were not considered above</td>
</tr>
</tbody>
</table>

- $n_1$: $V_\gamma = \{1, 2, 3\}$, $\text{NotEdge}_{\gamma \setminus \{2\}}$, $\text{NotEdge}_{\gamma \setminus \{1, 2\}}$, $\text{NotEdge}_{\gamma \setminus \{1, 3\}}$, $\text{NotEdge}_{\gamma \setminus \{2, 3\}}$, $\text{NotEdge}_{\gamma \setminus \{1, 2, 3\}}$
- $n_2$: $V_\gamma = \{3\}$, $\text{NotEdge}_{\gamma \setminus \emptyset}$, $\text{NotEdge}_{\gamma \setminus \{3\}}$, $\text{AdjInPar}_{\gamma \setminus \emptyset} = \{1, 3\}$
- $n_3$: $V_\gamma = \{2\}$, $\text{NotEdge}_{\gamma \setminus \emptyset}$, $\text{NotEdge}_{\gamma \setminus \{2\}}$, $\text{AdjInPar}_{\gamma \setminus \emptyset} = \{2\}$
- $n_4$: $V_\gamma = \{2\}$, $\text{NotEdge}_{\gamma \setminus \emptyset}$, $\text{NotEdge}_{\gamma \setminus \{2\}}$, $\text{AdjInPar}_{\gamma \setminus \emptyset} = \{1, 3\}$
- $n_5$: $V_\gamma = \{2, 3\}$, $\text{NotEdge}_{\gamma \setminus \{2\}}$, $\text{NotEdge}_{\gamma \setminus \{3\}}$, $\text{NotEdge}_{\gamma \setminus \{2, 3\}}$, $\text{Adj}(3) = \{1, 2\}$, $\text{Adj}(2) = \{1, 3\}$, $\text{AdjInPar}_{\gamma \setminus \{2\}} = \text{SameInPar}(\text{Adj}(3)) = \{3\}$, $\text{AdjInPar}_{\gamma \setminus \{3\}} = \{3\}$

The datalog solution. The datalog solution to the vertex-cover related evaluation problems follows immediately from the propositions of previous sections using the automaton defined above. More precisely, the solutions to VERTEX COVER, $k$-VERTEX COVER, MINIMUM VERTEX COVER CARDINALITY and MINIMUM VERTEX COVER are obtained via Propositions 4.2, 5.1, 5.2 and 5.3 respectively.
Table 8: The successful runs of “counting” automaton for Vertex Cover formula over $\mathcal{T}_{\mathcal{E},A}$.

<table>
<thead>
<tr>
<th>Successful runs</th>
<th>Sat. Assign.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_4 (a,b,c)$</td>
<td>$n_5 (c,d,e)$</td>
</tr>
<tr>
<td>$(\emptyset, Q^{(1)} , 0)$</td>
<td>$({2}, {3}, Q^{(1)}, 1)$</td>
</tr>
<tr>
<td>$(\emptyset, Q^{(1)}, 0)$</td>
<td>$({2}, {3}, Q^{(1)}, 1)$</td>
</tr>
<tr>
<td>$(\emptyset, Q^{(1)}, 0)$</td>
<td>$({2}, {3}, Q^{(1)}, 1)$</td>
</tr>
<tr>
<td>$(\emptyset, Q^{(1)}, 0)$</td>
<td>$({2}, {3}, Q^{(1)}, 1)$</td>
</tr>
<tr>
<td>$(\emptyset, Q^{(1)}, 0)$</td>
<td>$({2}, {3}, Q^{(1)}, 1)$</td>
</tr>
<tr>
<td>$(\emptyset, Q^{(1)}, 0)$</td>
<td>$({2}, {3}, Q^{(1)}, 1)$</td>
</tr>
</tbody>
</table>

| $(\{2\}, Q^{(1)}, 0)$ | $(\{2\}, \{3\}, Q^{(1)}, 1)$ | $(\{3\}, Q^{(1)}, 2)$ | $(\emptyset, Q^{(1)}, 0)$ | $(1, 2, 3), Q^{(0)}, 5$ | $\{d/e, c, a, g\}$ |
| $(\{2\}, Q^{(1)}, 0)$ | $(\{2\}, \{3\}, Q^{(1)}, 1)$ | $(\{3\}, Q^{(1)}, 2)$ | $(\emptyset, Q^{(1)}, 1)$ | $(1, 2, 3), Q^{(0)}, 6$ | $\{d/e, c, a, g, f\}$ |
| $(\{2\}, Q^{(1)}, 0)$ | $(\{2\}, \{3\}, Q^{(1)}, 1)$ | $(\{3\}, Q^{(1)}, 2)$ | $(\emptyset, Q^{(1)}, 1)$ | $(1, 2, 3), Q^{(0)}, 7$ | $\{d/e, c, a, g, f\}$ |
| $(\{2\}, Q^{(1)}, 0)$ | $(\{2\}, \{3\}, Q^{(1)}, 1)$ | $(\{3\}, Q^{(1)}, 2)$ | $(\emptyset, Q^{(1)}, 1)$ | $(1, 2, 3), Q^{(0)}, 8$ | $\{d/e, c, a, g, f\}$ |

29
Acknowledgments

We wish to thank Stavros Cosmadakis for valuable comments and suggestions.

References


