Monadic Datalog over Finite Structures of Bounded Treewidth

GEORG GOTTLOB
Oxford University
REINHARD PICHLER
Technische Universität Wien
and
FANG WEI
Albert-Ludwigs-Universität Freiburg

3

Bounded treewidth and monadic second-order (MSO) logic have proved to be key concepts in establishing fixed-parameter tractability results. Indeed, by Courcelle's Theorem we know that any property of finite structures, which is expressible by an MSO sentence, can be decided in linear time (data complexity) if the structures have bounded treewidth. In principle, Courcelle's Theorem can be applied directly to construct concrete algorithms by transforming the MSO evaluation problem into a tree language recognition problem. The latter can then be solved via a finite tree automaton (FTA). However, this approach has turned out to be problematical, since even relatively simple MSO formulae may lead to a "state explosion" of the FTA.

In this work we propose monadic datalog (i.e., datalog where all intentional predicate symbols are unary) as an alternative method to tackle this class of fixed-parameter tractable problems. We show that if some property of finite structures is expressible in MSO then this property can also be expressed by means of a monadic datalog program over the *decomposed structure*: we mean by this that the original structure is augmented with new elements and new relations that encode one of its tree decompositions. In the first place, we thus compare the expressive power of two query languages. However, we also show that the resulting fragment of datalog can be evaluated in linear time (both with respect to the program size and with respect to the data size). Hence, our transformation of an MSO query into a monadic datalog program yields an alternative proof of Courcelle's Theorem. In order to actually construct efficient algorithms for problems whose tractability is due to Courcelle's Theorem, we propose to use a fragment of full (i.e., not necessarily monadic) datalog which allows for a succinct representation of the corresponding monadic datalog

This is an extended and enhanced version of results published in Gottlob et al. [2007].
The work was supported by the Austrian Science Fund (FWF), project P20704-N18.
Authors' addresses: G. Gottlob, Computing Laboratory, Oxford University, Oxford OX1 3QD, U.K.; email: Georg.gottlob@comlab.ox.ac.uk; R. Pichler, Institut für Informationssysteme, Technische Universität Wien, A-1040 Vienna, Austria; email: pichler@dbai.tuwien.ac.at; F. Wei, Institut für Informatik, Albert-Ludwigs-Universität Freiburg, D-79110 Freiburg i. Br., Germany; email: fwei@informatik.uni-freiburg.de.

DOI 10.1145/1838552.1838555 http://doi.acm.org/10.1145/1838552.1838555

programs and for an efficient execution. This new approach is put to work by devising datalog programs for the 3-Colorability problem of graphs and for the PRIMALITY problem of relational schemas (i.e., testing if some attribute in a relational schema is part of a key). We also report on experimental results with a prototype implementation.

Categories and Subject Descriptors: F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Complexity of proof procedures, computations on discrete structures*; F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic—*Computational Logic*

General Terms: Algorithms, Performance, Theory

Additional Key Words and Phrases: Datalog, fixed-parameter tractability, Monadic second-order logic, tree decomposition, treewidth

ACM Reference Format:

Gottlob, G., Pichler, R., and Wei, F. 2010. Monadic datalog over finite structures of bounded treewidth. ACM Trans. Comput. Logic, 12, 1, Article 3 (October 2010), 48 pages.
DOI = 10.1145/1838552.1838555 http://doi.acm.org/10.1145/1838552.1838555

1. INTRODUCTION

Over the past decade, parameterized complexity has evolved as an important subdiscipline in the field of computational complexity (see Downey and Fellows [1999]; Flum and Grohe [2006]). In particular, it has been shown that many hard problems become tractable if some problem parameter is fixed or bounded by a constant. In the arena of graphs and, more generally, of finite structures, the treewidth is one such parameter which has served as the key to many fixed-parameter tractability (FPT) results. The most prominent method for establishing the FPT in case of bounded treewidth is via Courcelle's Theorem (see Courcelle [1990a, 1990b]): any property of finite structures, which is expressible by a monadic second-order (MSO) sentence, can be decided in linear time (data complexity) if the treewidth of the structures is bounded by a fixed constant.

Recipes as to how one can devise concrete algorithms based on Courcelle's Theorem can be found in the literature (see Flum et al. [2002], which is based upon earlier work like Arnborg et al. [1991]). The idea is to first translate the MSO evaluation problem over finite structures into an equivalent MSO evaluation problem over rooted, colored binary trees. This problem can then be solved via the correspondence between MSO over terms and finite tree automata (FTA) (see Thatcher and Wright [1968]; Doner [1970]; Thomas [1997]) (note that labeled, rooted trees of bounded degree can be seen as terms). In theory, this generic method of turning an MSO description into a concrete algorithm looks very appealing. However, in practice, it has turned out that even relatively simple MSO formulae may lead to a "state explosion" of the FTA (see Frick and Grohe [2004]; Maryns [2006]). Consequently, it was already noted in Grohe [1999] that the algorithms derived via Courcelle's Theorem are useless for practical applications. The main benefit of Courcelle's Theorem is that it provides a simple way to recognize a property as being linear time computable. In other words, proving the FPT of some problem by showing that it is MSO expressible is the starting point (rather than the end point) of the search for an efficient algorithm.

In this work we investigate the potential of monadic datalog (i.e., datalog where all intensional predicate symbols are unary) for devising efficient algorithms in situations where the FPT has been shown via Courcelle's Theorem. Above all, we prove that if some property of finite structures is expressible in MSO then this property can also be expressed by means of a monadic datalog program over the *decomposed structure*: we mean by this that the original structure is augmented with new elements and new relations that encode one of its tree decompositions. Hence, in the first place, we prove an *expressivity result* rather than a mere complexity result. However, we also show that the resulting fragment of datalog can be evaluated in linear time (both with respect to the program size and with respect to the data size). We thus get the corresponding *complexity result* (i.e., Courcelle's Theorem) as a corollary of this MSO-to-datalog transformation.

Our MSO-to-datalog transformation for finite structures of bounded treewidth generalizes a result from Gottlob and Koch [2004], where it was shown that MSO on trees has the same expressive power as monadic datalog on trees. Note that the connection between logic programming (with functions) and tree automata has already been studied much earlier (see, e.g., Marque-Pucheu [1983]; Filé [1985]). Several obstacles had to be overcome to prove our generalization.

—First of all, we no longer have to deal with a single universe, namely, the universe of trees whose domain consists of the tree nodes. Instead, we now have to deal with—and constantly switch between—two universes, namely, the relational structure (with its own signature and its own domain), on the one hand, and the tree decomposition (with appropriate predicates expressing the tree structure and with the tree nodes as a separate domain), on the other.

—Of course, not only the MSO-to-datalog transformation itself had to be lifted to the case of two universes. Also important prerequisites of the results in Gottlob and Koch [2004] (notably several results on MSO-equivalences of tree structures shown in Neven and Schwentick [2002]) had to be extended to this new situation.

—Apart from switching between the two universes, it is ultimately necessary to integrate both universes into the monadic datalog program. For this purpose, both the signature and the domain of the finite structure have to be appropriately extended.

—It has turned out that previous notions of standard or normal forms of tree decompositions (see Downey and Fellows [1999]; Flum et al. [2002]) are not suitable for our purposes. We therefore have to introduce a modified version of "normalized tree decompositions," which is then further refined as we present new algorithms based on the monadic datalog approach.

In the second part of this article, we show how the monadic datalog approach can be used to devise efficient algorithms. To this end, we use a fragment of full (i.e., not necessarily monadic) datalog which allows for a succinct representation of the corresponding monadic datalog programs and for an efficient execution. We put this approach to work by presenting datalog programs for

the 3-Colorability problem of graphs and for the PRIMALITY problem of relational schemas (i.e., testing if some attribute in a relational schema is part of a key). Both problems are well known to be intractable (see Beeri and Bernstein [1979]; Mannila and Räihä [1992] for PRIMALITY). It is folklore that the 3-Colorability problem can be expressed by an MSO sentence. In Gottlob et al. [2006b], it was shown that PRIMALITY is MSO expressible. Hence, in the case of bounded treewidth, both problems become tractable. However, two attempts to tackle these problems via the standard MSO-to-FTA approach turned out to be very problematical: we experimented with a prototype implementation using MONA (see Klarlund et al. [2002]) for the MSO model checking, but we ended up with "out-of-memory" errors already for really small input data (see Section 6). Alternatively, we made an attempt to directly implement the MSO-to-FTA mapping proposed in Flum et al. [2002]. However, the "state explosion" of the resulting FTA—which tends to occur already for comparatively simple formulae (cf. Maryns [2006])—led to failure before we were able to feed any input data to the program.

In contrast, the experimental results with our new datalog approach look very promising, see Section 6. By the experience gained with these experiments, the following advantages of datalog compared with MSO became apparent:

—*Level of declarativity*. MSO as a logic has the highest level of declarativity, which often allows one very elegant and succinct problem specifications. However, MSO does not have an operational semantics. In order to turn an MSO specification into an algorithm, the standard approach is to transform the MSO evaluation problem into a tree language recognition problem. But the FTA clearly has a much lower level of declarativity and the intuition of the original problem is usually lost when an FTA is constructed. In contrast, the datalog program with its declarative style often reflects both the intuition of the original problem and of the algorithmic solution. This intuition can be exploited for defining heuristics which lead to problem-specific optimizations.

—*General optimizations*. A lot of research has been devoted to generally applicable (i.e., not problem-specific) optimization techniques of datalog (see, e.g., Ceri et al. [1990]). In our implementation (see Section 6), we make heavy use of these optimization techniques, which are not available in the MSO-to-FTA approach.

—*Flexibility*. The generic transformation of MSO formulae to monadic datalog programs (given in Section 4) inevitably leads to programs of exponential size with respect to the size of the MSO-formula and the treewidth. However, as our programs for 3-Colorability and PRIMALITY demonstrate, many relevant properties can be expressed by really short programs if we allow nonmonadic datalog. Moreover, as we will see in Section 5, also datalog provides us with a certain level of succinctness. In fact, we will be able to express a big monadic datalog program by a small nonmonadic program, whose size can be even further reduced by allowing set operations in the datalog programs.

—*Required transformations*. The problem of a "state explosion" reported in Maryns [2006] already refers to the transformation of (relatively simple) MSO formulae *on trees* to an FTA. If we consider MSO *on structures of*

bounded treewidth, the situation gets even worse, since the original (possibly simple) MSO formula over a finite structure first has to be transformed into an equivalent MSO formula over trees. This transformation (e.g., by the algorithm in Flum et al. [2002]) leads to a much more complex formula (in general, even with additional quantifier alternations) than the original formula. In contrast, our approach works with monadic datalog programs on finite structures which need no further transformation. Each program can be executed as it is.

—*Extending the programming language*. One more aspect of the flexibility of datalog is the possibility to define new built-in predicates, which expose an efficient implementation in some other programming language (typically in an imperative language) via a logical predicate. This is a standard technique used by Prolog systems (which typically provide built-in predicates for type testing, term unification, term comparison, arithmetic, input/output, etc.) and it is also applicable to datalog. For instance, dlvhex [Eiter et al. 2005, 2006] allows the user to plug in program modules written in an imperative programming language into a datalog program that is executed by the dlv system [Leone et al. 2006]. Another example of a useful language extension is the introduction of generalized quantifiers, which allow us to modularly add features which are only expressible in higher-order logic. For instance, Härtig quantifiers (by which we can express equicardinality) can thus be represented. Likewise, Henkin quantifiers [Herre et al. 1991], which have interesting applications to linguistics [Sher 1997], fall into this category. For the theoretical background of generalized quantifiers, see Eiter et al. [1997a, 1997b] and Gottlob [1997].

Some applications require a fast execution which cannot always be guaranteed by an interpreter. Hence, while we propose a logic programming approach, one can of course go one step further and implement our algorithms directly in Java, C++, etc., following the same paradigm.

The article is organized as follows. After recalling some basic notions and results in Section 2, we present several results on the MSO equivalence of substructures induced by subtrees of a tree decomposition in Section 3. In Section 4, it is shown that any MSO formula with one free individual variable over structures of bounded treewidth can be transformed into an equivalent monadic datalog program. In Section 5, we put the monadic datalog approach to work by presenting datalog programs for the 3-Colorability problem and for the PRIMALITY problem in the case of bounded treewidth. In Section 6, we report on experimental results with a prototype implementation. A conclusion is given in Section 7.

2. PRELIMINARIES

2.1 Relational Schemas and Primality

We briefly recall some basic notions and results from database design theory (for details, see Mannila and Räihä [1992]). In particular, we shall define the

PRIMALITY problem, which will serve as a running example throughout this article.

A relational schema is denoted as (R, F), where R is the set of attributes, and F the set of functional dependencies (FDs, for short) over R. Without loss of generality, we only consider FDs whose right-hand side consists of a single attribute. Let $f \in F$ with $f: Y \rightarrow A$. We refer to $Y \subseteq R$ and $A \in R$ as $lhs(f)$ and $rhs(f)$, respectively. The intended meaning of an FD $f: Y \rightarrow A$ is that, in any valid database instance of (R, F), the value of the attribute A is uniquely determined by the value of the attributes in Y. It is convenient to denote a set $\{A_1, A_2, \ldots, A_n\}$ of attributes as a string $A_1A_2 \ldots A_n$. For instance, we write $f: ab \rightarrow c$ rather than $f: \{a, b\} \rightarrow c$.

For any $X \subseteq R$, we write X^+ to denote the closure of X, that is, the set of all attributes determined by X. An attribute A is contained in X^+ if and only if either $A \in X$ or there exists a "derivation sequence" of A from X in F of the form $X \rightarrow X \cup \{A_1\} \rightarrow X \cup \{A_1, A_2\} \rightarrow \cdots \rightarrow X \cup \{A_1, \ldots, A_n\}$, such that $A_n = A$ and for every $i \in \{1, \ldots, n\}$, there exists an FD $f_i \in F$ with $lhs(f) \subseteq X \cup \{A_1, \ldots, A_{i-1}\}$ and $rhs(f) = A_i$.

If $X^+ = R$ then X is called a *superkey*. If X is minimal with this property, then X is a *key*. An attribute A is called *prime* if it is contained in at least one key in (R, F). An efficient algorithm for testing the primality of an attribute is crucial in database design since it is an indispensable prerequisite for testing if a schema is in third normal form. However, given a relational schema (R, F) and an attribute $A \in R$, it is NP-complete to test if A is prime (cf. Beeri and Bernstein [1979]; Mannila and Räihä [1992]). Clearly, such computations are independent of the concrete data and are done once and for all when designing a relational database. Hence, time efficiency is less critical than in the case of actual data access. Nevertheless, the intractability of certain computational problems was identified in Beeri and Bernstein [1979] as a serious obstacle to a good database design. Recognizing primality is one such problem.

We shall consider two variants of the PRIMALITY problem in this article (see Sections 5.2 and 5.3, respectively): the decision problem (i.e, given a relational schema (R, F) and an attribute $A \in R$, is A prime in (R, F)?) and the enumeration problem (i.e, given a relational schema (R, F), compute all prime attributes in (R, F)).

Example 2.1. Consider the relational schema (R, F) with $R = abcdeg$ and $F = \{f_1: ab \rightarrow c,\ f_2: c \rightarrow b,\ f_3: cd \rightarrow e,\ f_4: de \rightarrow g,\ f_5: g \rightarrow e\}$. It can be easily checked that there are two keys for the schema: abd and acd. Thus, the attributes a, b, c, and d are prime, while e and g are not.

2.2 Finite Structures and Treewidth

All structures and trees considered in this work are assumed to be *finite*. Let $\tau = \{R_1, \ldots, R_K\}$ be a set of predicate symbols. A *(finite) structure* $\mathcal{A}$ over τ (a *τ-structure*, for short) is given by a *finite* domain $A = dom(\mathcal{A})$ and relations $R_i^{\mathcal{A}} \subseteq A^{\alpha}$, where α denotes the arity of $R_i \in \tau$. A structure may also be given in the form $(\mathcal{A}, \bar{a})$ where, in addition to $\mathcal{A}$, we have distinguished elements

$\bar{a} = (a_0, \ldots, a_w)$ from $dom(\mathcal{A})$. Such distinguished elements are required for interpreting formulae with free variables.

We write $|\mathcal{A}|$ to denote the size of (a "reasonable" encoding of) a structure. Following Flum et al. [2002], a structure can be encoded in the standard RAM model by first encoding the signature τ and then encoding the domain A and each realation $R_i^{\mathcal{A}}$. The encoding of $R_i^{\mathcal{A}}$ simply consists in specifying all α-tuples in $R_i^{\mathcal{A}}$, where α denotes the arity of R_i. Hence, for the size $|R_i^{\mathcal{A}}|$ of $R_i^{\mathcal{A}}$, we get $|R_i^{\mathcal{A}}| = \Theta(\alpha \cdot card(R_i^{\mathcal{A}}))$, where $card(R_i^{\mathcal{A}})$ denotes the number of tuples in $R_i^{\mathcal{A}}$. In total, the size of a τ-structure $\mathcal{A}$ is $\Theta(|\tau| + |A| + \sum_{R_i \in \tau} |R_i^{\mathcal{A}}|)$.

A *tree decomposition* $\mathcal{T}$ of a τ-structure $\mathcal{A}$ is defined as a pair $\langle T, (A_t)_{t \in T} \rangle$ where T is a tree and each A_t is a subset of A with the following properties: (1) every $a \in A$ is contained in some A_t. (2) For every $R_i \in \tau$ and every tuple $(a_1, \ldots, a_\alpha) \in R_i^{\mathcal{A}}$, there exists some node $t \in T$ with $\{a_1, \ldots, a_\alpha\} \subseteq A_t$. (3) For every $a \in A$, the set $\{t \mid a \in A_t\}$ induces a subtree of T.

Without loss of generality, we assume that the tree T underlying a tree decomposition $\mathcal{T} = \langle T, (A_t)_{t \in T} \rangle$ is rooted and has bounded degree. Indeed, considering some node of T as the root has no effect on the above definition of tree decompositions. Moreover, the restriction to bounded degree can be easily achieved (below, we shall show that, without loss of generality, we may even restrict ourselves to *binary* trees; see Proposition 2.4). These assumptions on the tree T play an important role for the equivalence of MSO and automata (see Section 2.3). The classical equivalence between MSO and automata works for automata on *terms*. However, since labeled, rooted trees of bounded degree can be seen as terms, we thus get the equivalence between MSO and automata on *trees*.

The third condition in the above definition of tree decompositions is usually referred to as the *connectedness condition*. The sets A_t are called the *bags* (or *blocks*) of $\mathcal{T}$. The *width* of a tree decomposition $\langle T, (A_t)_{t \in T} \rangle$ is defined as $max\{|A_t| \mid t \in T\} - 1$. The *treewidth* of $\mathcal{A}$ is the minimal width of all tree decompositions of $\mathcal{A}$. It is denoted as $tw(\mathcal{A})$. Note that trees and forests are precisely the structures of treewidth 1.

For given $w \geq 1$, it can be decided in linear time if some structure has treewidth at most w. Moreover, in case of a positive answer, a tree decomposition of width w can be computed in linear time; see Bodlaender [1996]. Strictly speaking, the result in Bodlaender [1996] refers to tree decompositions of *graphs* rather than arbitrary *structures*. However, we can associate a graph $\mathcal{G}$ (the so-called primal or Gaifman graph) with every structure $\mathcal{A}$ by taking the domain elements as the vertices of the graph. Moreover, two vertices are adjacent in $\mathcal{G}$ if and only if the corresponding domain elements jointly occur in some tuple in $\mathcal{A}$. It can be easily shown that $\mathcal{G}$ has precisely the same tree decompositions as $\mathcal{A}$.

Unfortunately, it has been shown that the linear time algorithm from Bodlaender [1996] is mainly of theoretical interest and the practical usefulness is limited [Koster et al. 2001]. Recently, considerable progress has been made in developing heuristic-based tree decomposition algorithms which can handle graphs of moderate size with several hundreds of vertices [Koster et al. 2001;

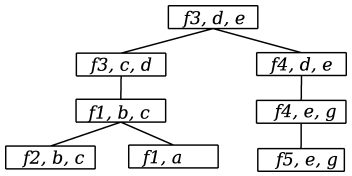

Fig. 1. Tree decomposition $\mathcal{T}$ of schema (R, F) in Example 2.1.

Bodlaender and Koster 2006, 2008; van den Eijkhof et al. 2007]. Moreover, in some cases, the tree decomposition may be obtained from the origin of the problem, that may be structured in a "natural way." For instance, in Thorup [1998], it was shown that the tree width of the control-flow graph of any goto-free C program is at most six. A similar result was shown for Java programs in Gustedt et al. [2002]. These results opened the ground for the efficient implementation of various compiler optimization tasks like the register allocation problem.

In this article, we assume that a relational schema (R, F) is given as a τ-structure with $\tau = \{fd, att, lh, rh\}$. The intended meaning of these predicates is as follows: $fd(f)$ means that f is an FD and $att(b)$ means that b is an attribute. $lh(b, f)$ (respectively, $rh(b, f)$) means that b occurs in $lhs(f)$ (respectively, in $rhs(f)$). The treewidth of (R, F) is then defined as the treewidth of this τ structure.

Example 2.2. Recall the relational schema (R, F) with $R = abcdeg$ and $F = \{f_1{:}ab \to c,\ f_2{:}c \to b,\ f_3{:}cd \to e,\ f_4{:}de \to g,\ f_5{:}g \to e\}$ from Example 2.1. This schema is represented as the following τ-structure with $\tau = \{fd, att, lh, rh\}$: $\mathcal{A} = (A, fd^{\mathcal{A}}, att^{\mathcal{A}}, lh^{\mathcal{A}}, rh^{\mathcal{A}})$ with $A = R$, $fd^{\mathcal{A}} = \{f_1, f_2, f_3, f_4, f_5\}$, $att^{\mathcal{A}} = \{a, b, c, d, e, g\}$, $lh^{\mathcal{A}} = \{(a, f_1), (b, f_1), (c, f_2), (c, f_3), (d, f_3), (d, f_4), (e, f_4), (g, f_5)\}$, $rh^{\mathcal{A}} = \{(c, f_1), (b, f_2), (e, f_3), (g, f_4), (e, f_5)$.

A tree decomposition $\mathcal{T}$ of this structure is given in Figure 1. Note that the maximal size of the bags in $\mathcal{T}$ is 3. Hence, the tree width is at most 2. On the other hand, it is easy to check that the tree-width of $\mathcal{T}$ cannot be smaller than 2: in order to see this, we consider the tuples in $lh^{\mathcal{A}}$ and $rh^{\mathcal{A}}$ as edges of an undirected graph. Then the edges corresponding to $(b, f_1), (c, f_2) \in lh^{\mathcal{A}}$ and $(b, f_2), (c, f_1) \in rh^{\mathcal{A}}$ form a cycle in this graph. However, as we have recalled above, only trees and forests have treewidth 1. The tree decomposition in Figure 1 is, therefore, optimal and we have $tw(F) = tw(\mathcal{A}) = 2$.

Remark. A relational schema (R, F) defines a hypergraph $H(R, F)$ whose vertices are the attributes in R and whose hyperedges are the sets of attributes jointly occurring in at least one FD in F. Recall that the incidence graph of a hypergraph H contains as nodes the vertices and hyperedges of H. Moreover, two nodes v and h (corresponding to a vertex v and a hyperedge h in H) are connected in this graph if and only if (in the hypergraph H) v occurs in h. It can be easily verified that the treewidth of the above described τ-structure and of the incidence graph of the hypergraph $H(R, F)$ coincide.

In Section 4, it is convenient to consider the elements in the bags of a tree decomposition as ordered. Similarly to the normal form introduced in Theorem 6.72 of Downey and Fellows [1999], we will use the following form of *normalized tree decompositions*.

Definition 2.3. Let $\mathcal{A}$ be a structure with tree decomposition $\mathcal{T}$ of width w. We call $\mathcal{T}$ *normalized* if the following conditions are fulfilled:

(1) The bags are considered as tuples of $w+1$ pairwise distinct elements $(a_0, \ldots, a_w)$ rather than sets.
(2) Every internal node $t \in T$ has either 1 or 2 child nodes.
(3) If a node t with bag $(a_0, \ldots, a_w)$ has one child node, then the bag of the child is either obtained via a permutation of $(a_0, \ldots, a_w)$ or by replacing a_0 with another element a_0'. We call such a node t a *permutation node* or an *element replacement node*, respectively.
(4) If a node t has two child nodes then these child nodes have identical bags as t. In this case, we call t a *branch node*.

Finally, we also request that the number of nodes of the tree be linear in the number of domain elements in $\mathcal{A}$. In other words, we forbid "useless permutations." Several successive permutations can be fused into a single one.

PROPOSITION 2.4. *Let $\mathcal{A}$ be a structure with tree decomposition $\mathcal{T}$ of width w. Without loss of generality, we may assume that the domain $dom(\mathcal{A})$ has at least $w+1$ elements. Then $\mathcal{T}$ can be transformed in linear time into a normalized tree decomposition $\mathcal{T}'$, such that $\mathcal{T}$ and $\mathcal{T}'$ have identical width.*

PROOF. We can transform an arbitrary tree decomposition $\mathcal{T}$ into a normalized tree decomposition $\mathcal{T}'$ by the following steps (1)–(5). Clearly this transformation works in linear time and preserves the width.

(1) All bags can be padded to the "full" size of $w+1$ elements by adding elements from a neighboring bag. For example, let s and s' be adjacent nodes and let A_s have $w+1$ elements (in a tree decomposition of width w, at least one such node exists) and let $|A_{s'}| = w'+1$ with $w' < w$. Then $|A_s \setminus A_{s'}| \geq (w - w')$ and we may simply add $(w - w')$ elements from $A_s \setminus A_{s'}$ to $A_{s'}$ without violating the connectedness condition.
(2) Suppose that some internal node s has $k+2$ child nodes $t_1, \ldots, t_{k+2}$ with $k > 0$. It is a standard technique to turn this part of the tree into a binary tree by inserting copies of s into the tree, that is, we introduce k nodes $s_1, \ldots, s_k$ with $A_{s_i} = A_s$, such that the second child of s is s_1, the second child of s_1 is s_2, the second child of s_2 is s_3, etc. Moreover, t_1 remains the first child of s, while t_2 becomes the first child of s_1, t_3 becomes the first child of $s_2, \ldots,$ t_{k+1} becomes the first child of s_k. Finally, t_{k+2} becomes the second child of s_k. Clearly, the connectedness condition is preserved by this construction.
(3) If an internal node s has two children t_1 and t_2, such that the bags of s, t_1, and t_2 are not identical, then we simply insert a copy s_1 of s between s and t_1 and another copy s_2 of s between s and t_2.

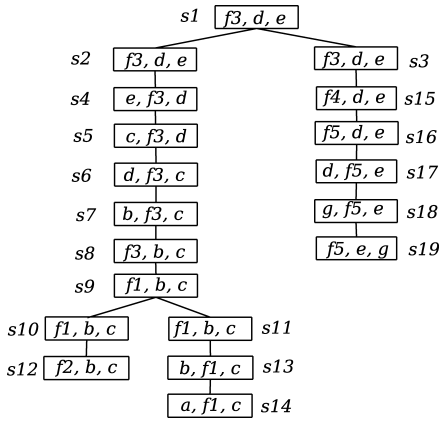

Fig. 2. Normalized tree decomposition $\mathcal{T}'$ of schema (R, F) in Example 2.1.

(4) Let s be the parent of s' and let $|A_s \setminus A_{s'}| = k$ with $k > 1$. Then we can obviously "interpolate" s and s' by new nodes $s_1, \ldots, s_{k-1}$, such that s_{k-1} is the new parent of s', s_{k-2} is the parent of s_{k-1}, $\ldots$, s is the parent of s_1. Moreover, the bags A_{s_i} can be defined in such a way that the bags of any two neighboring nodes differ in exactly one element, for example, $|A_s \setminus A_{s_1}| = |A_{s_1} \setminus A_s| = 1$.

(5) Let the bags of any two neighboring nodes s and s' differ by one element, that is, $\exists a \in A_s$ with $a \notin A_{s'}$ and $\exists a' \in A_{s'}$ with $a' \notin A_s$. Then we can insert two "interpolation nodes" t and t', such that A_t has the same elements as A_s but with a at position 0. Likewise, $A_{t'}$ has the same elements as $A_{s'}$ but with a' at position 0. □

Example 2.5. The tree decomposition $\mathcal{T}$ in Figure 1 is clearly not normalized. In contrast, the tree decomposition $\mathcal{T}'$ in Figure 2 is normalized in the above sense. Let us ignore the node identifiers $s_1, \ldots, s_{19}$ for the moment. Note that $\mathcal{T}$ and $\mathcal{T}'$ have identical width.

When we devise concrete datalog programs for the 3-Colorability problem of graphs and for the PRIMALITY problem of relational schemas in Section 5, it is preferable to return to the notion of tree decompositions whose bags are *sets* of domain elements rather than tuples. Hence, we may delete permutation nodes from the tree decomposition. Moreover, it is convenient to split the action of *element replacement nodes* in two steps. Recall that an element replacement node replaces exactly one element in the bag of the child node by a new element. In our algorithms in Section 5, we shall replace these *element replacement nodes* by two new kinds of nodes, namely, *element removal nodes* (which remove one domain element from the bag of the child node) and *element introduction nodes* (which introduce one new element). Finally, we drop the condition that all bags in a tree decomposition of width w must have "full size" $w + 1$ (by splitting the element replacement into element removal and element introduction, this condition would have required some relaxation anyway). In summary, we get the following modified normal form, which was also considered in Kloks [1994].

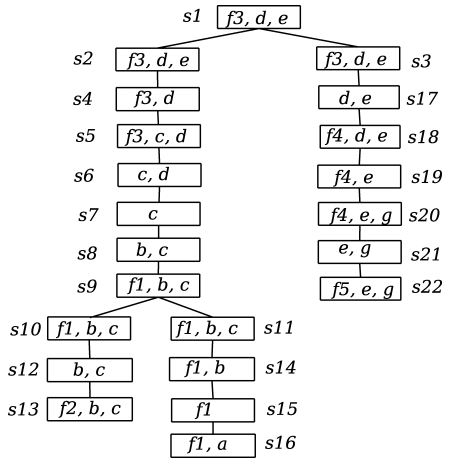

Fig. 3. Normalized tree decompositions with set bags.

Definition 2.6. Let $\mathcal{A}$ be an arbitrary structure with tree decomposition $\mathcal{T}$ of width w. We call $\mathcal{T}$ *normalized with set bags* if the following conditions are fulfilled:

(1) The bags are considered as sets of at most $k+1$ pairwise distinct elements $\{a_0, \ldots, a_k\}$ with $k \leq w$.
(2) Every internal node $t \in T$ has either 1 or 2 child nodes.
(3) If a node t with bag $\{a_0, \ldots, a_k\}$ has one child node, then the bag of the parent is obtained from the bag of the child either by introducing a new element or by removing one element. We call such a node t an *element introduction node* or an *element removal node*, respectively.
(4) If a node t has two child nodes then these child nodes have identical bags as t. In this case, we call t a *branch node*.

Finally, we again request that the number of nodes of the tree be linear in the number of domain elements in $\mathcal{A}$. In other words, we forbid "useless copies" of bags.

Clearly, every tree decomposition $\mathcal{T}$ can be transformed in linear time into a normalized tree decomposition $\mathcal{T}''$ with set bags according to Definition 2.6, such that $\mathcal{T}$ and $\mathcal{T}''$ have identical width. For instance, recall the tree decomposition $\mathcal{T}'$ from Figure 2. A tree decomposition $\mathcal{T}''$ compliant with our *normal form with set bags* from Definition 2.6 is depicted in Figure 3.

2.3 Monadic Second-Order Logic

We assume some familiarity with monadic second-order logic (MSO) (see, e.g., Ebbinghaus and Flum [1999]; Libkin [2004]). MSO extends first order (FO) logic by the use of *set variables* (usually denoted by uppercase letters), which range over sets of domain elements. In contrast, the *individual variables* (which are usually denoted by lowercase letters) range over single domain elements. An FO formula φ over a τ-structure has as atomic formulae either atoms with

some predicate symbol from τ or equality atoms. An MSO formula φ over a τ-structure may additionally have atoms whose predicate symbol is a monadic predicate variable. For the sake of readability, we denote such an atom usually as $a \in X$ rather than $X(a)$. Likewise, we use set operators $\subseteq$ and $\subset$ with the obvious meaning.

The *quantifier depth* of an MSO formula φ is defined as the maximum degree of nesting of quantifiers (both for individual variables and set variables) in φ. In this work, we will mainly encounter MSO formulae with free individual variables. A formula $\varphi(x)$ with exactly one free individual variable is called a *unary query*. More generally, let $\varphi(\bar{x})$ with $\bar{x} = (x_0, \ldots, x_w)$ for some $w \geq 0$ be an MSO formula with free variables $\bar{x}$. Furthermore, let $\mathcal{A}$ be a τ-structure and $\bar{a} = (a_0, \ldots, a_w)$ be distinguished domain elements. We write $(\mathcal{A}, \bar{a}) \models \varphi(\bar{x})$ to denote that $\varphi(\bar{a})$ evaluates to true in $\mathcal{A}$. Usually, we refer to $(\mathcal{A}, \bar{a})$ simply as a *structure* rather than a *structure with distinguished domain elements*.

Example 2.7. It was shown in Gottlob et al. [2006b] that primality can be expressed in MSO. We give a slightly different MSO formula $\varphi(x)$ here, which is better suited for our purposes in Section 5, namely,

$\varphi(x) = (\exists Y)[Y \subseteq R \wedge \mathit{Closed}(Y) \wedge x \notin Y \wedge \mathit{Closure}(Y \cup \{x\}, R)]$ with
$\mathit{Closed}(Y) \equiv (\forall f)[fd(f) \rightarrow (\exists b)[(rh(b, f) \wedge b \in Y) \vee (lh(b, f) \wedge b \notin Y)]]$ and
$\mathit{Closure}(Y, Z) \equiv Y \subseteq Z \wedge \mathit{Closed}(Z) \wedge \neg(\exists Z')[Y \subseteq Z' \wedge Z' \subset Z \wedge \mathit{Closed}(Z')]$.

This formula expresses the following characterization of primality: an attribute a is prime if and only if there exists an attribute set $\mathcal{Y} \subseteq R$, such that $\mathcal{Y}$ is closed with respect to F (i.e., $\mathcal{Y}^+ = \mathcal{Y}$), $a \notin \mathcal{Y}$, and $(\mathcal{Y} \cup \{a\})^+ = R$. In other words, $\mathcal{Y} \cup \{a\}$ is a superkey but $\mathcal{Y}$ is not.

Recall the τ-structure $\mathcal{A}$ from Example 2.2 representing a relational schema. It can be easily verified that $(\mathcal{A}, a) \models \varphi(x)$ and $(\mathcal{A}, e) \not\models \varphi(x)$ hold. Clearly, the quantifier depth of the formulae $\mathit{Closed}(Y)$ is 2. As far as the formula $\mathit{Closure}(Y, Z)$ is concerned, we have to first consider the quantifier depth of the subformulae $Y \subseteq Z$ and $Y \subset Z$, which can be expressed as $\forall x[x \in Y \rightarrow x \in Z]$, and $Y \subseteq Z \wedge \exists x[x \in Z \wedge x \notin Y]$, respectively. Thus the quantifier depth of both subformulae is 1. Therefore, the quantifier depths of the formula $\mathit{Closure}(Y, Z)$ and $\varphi(x)$ are 3 and 4, respectively.

We call two structures $(\mathcal{A}, \bar{a})$ and $(\mathcal{B}, \bar{b})$ *k-equivalent* and write $(\mathcal{A}, \bar{a}) \equiv_k^{MSO} (\mathcal{B}, \bar{b})$ if and only if, for every MSO formula φ of quantifier depth at most k, the equivalence $(\mathcal{A}, \bar{a}) \models \varphi \Leftrightarrow (\mathcal{B}, \bar{b}) \models \varphi$ holds. By definition, $\equiv_k^{MSO}$ is an equivalence relation. For any k, the relation $\equiv_k^{MSO}$ has only finitely many equivalence classes. These equivalence classes are referred to as *k-types* or simply as *types*. The $\equiv_k^{MSO}$ equivalence between two structures can be effectively decided. There is a nice characterization of $\equiv_k^{MSO}$-equivalence by Ehrenfeucht-Fraïssé games: The *k-round MSO game* on two structures $(\mathcal{A}, \bar{a})$ and $(\mathcal{B}, \bar{b})$ is played between two players—the spoiler and the duplicator. In each of the k rounds, the spoiler can choose between a point move and a set move. If, in the ith round, the spoiler makes a *point move*, he or she then selects some element $c_i \in dom(\mathcal{A})$ or some element $d_i \in dom(\mathcal{B})$. The duplicator answers by choosing an element in the opposite structure. If, in the ith round, the spoiler makes a *set move*, he or she

then selects a set $P_i \subseteq dom(\mathcal{A})$ or a set $Q_i \subseteq dom(\mathcal{B})$. The duplicator answers by choosing a set of domain elements in the opposite structure. Suppose that, in k rounds, the domain elements $c_1, \ldots, c_m$ and $d_1, \ldots, d_m$ from $dom(\mathcal{A})$ and $dom(\mathcal{B})$, respectively, were chosen in the point moves. Likewise, suppose that the subsets $P_1, \ldots, P_n$ and $Q_1, \ldots, Q_m$ of $dom(\mathcal{A})$ and $dom(\mathcal{B})$, respectively, were chosen in the set moves. The *duplicator wins* this game, if the mapping which maps each c_i to d_i is a partial isomorphism from $(\mathcal{A}, \bar{a}, P_1, \ldots, P_n)$ to $(\mathcal{B}, \bar{b}, Q_1, \ldots, Q_n)$. We say that the duplicator has a *winning strategy* in the k-round MSO game on $(\mathcal{A}, \bar{a})$ and $(\mathcal{B}, \bar{b})$ if he or she can win the game for any possible moves of the spoiler.

The following relationship between $\equiv_k^{MSO}$ equivalence and k-round MSO games holds: *Two structures* $(\mathcal{A}, \bar{a})$ *and* $(\mathcal{B}, \bar{b})$ *are k-equivalent if and only if the duplicator has a winning strategy in the k-round MSO game on* $(\mathcal{A}, \bar{a})$ *and* $(\mathcal{B}, \bar{b})$, (see Ebbinghaus and Flum [1999]; Libkin [2004]).

The importance of MSO in the context of parameterized complexity comes from Courcelle's Theorem, which can be phrased as follows:

THEOREM 2.8 (COURCELLE; 1987 1990A, 1990B). *The model checking problem for an MSO sentence* φ *and a* τ*-structure* $\mathcal{A}$ *is fixed-parameter linear with parameter* $(|\varphi|, tw(\mathcal{A}))$*, that is, checking if an MSO sentence* φ *evaluates to true over a* τ*-structure* $\mathcal{A}$ *of treewidth* w *can be done in time* $\mathcal{O}(f(|\varphi|, w) * |\mathcal{A}|)$ *for some function* f.

It is important to note that the fixed-parameter linearity according to the above theorem does not immediately guarantee practical algorithms due to the huge "hidden" constants (which are nonelementary in general). Courcelle's Theorem can be proved in several ways. The proofs in [Courcelle 1987, 1990b] were based on the Feferman Vaught Theorem (see also Makowsky [2004]). Alternative proofs are based on the relationship between MSO and finite-tree automata (see e.g., Arnborg et al. [1991]; Flum et al. [2002]) or Ehrenfeucht-Fraïssé games (as is our proof in Section 4). In any case, we end up with constants that are nonelementary with respect to some parameter of the MSO formula, namely, the number of quantifier alternations (in the case of the automata theoretic approach) or the quantifier depth (in the case of the other two approaches).

2.4 Datalog

We assume some familiarity with datalog (see, e.g., Abiteboul et al. [1995]; Ceri et al. [1990]; Ullman [1989]). Syntactically, a datalog program $\mathcal{P}$ is a set of function-free, definite Horn clauses, that is, each clause consists of a nonempty head and a possibly empty body. Clauses with nonempty bodies are called *rules* while those with empty bodies are called *facts*. Predicates occurring only in the body of rules in $\mathcal{P}$ are called *extensional*, while predicates occurring also in the head of some rule are called *intensional*.

Let $\mathcal{A}$ be a τ-structure with domain A and relations $R_1^{\mathcal{A}}, \ldots, R_K^{\mathcal{A}}$ with $R_i^{\mathcal{A}} \subseteq A^{\alpha}$, where α denotes the arity of $R_i \in \tau$. In the context of datalog, it is convenient to think of the relations $R_i^{\mathcal{A}}$ as sets of ground atoms. The set of all such ground

atoms of a structure $\mathcal{A}$ is referred to as the *extensional database* (EDB) of $\mathcal{A}$, which we shall denote as $\mathcal{E}(\mathcal{A})$ (or simply as $\mathcal{A}$, if no confusion is possible). We have $R_i(\bar{a}) \in \mathcal{E}(\mathcal{A})$ if and only if $\bar{a} \in R_i^{\mathcal{A}}$.

In order to evaluate a datalog program $\mathcal{P}$ over a structure $\mathcal{A}$, we consider the atoms in $\mathcal{A}$ as additional facts of the program. The result of this evaluation is the set of those (ground) facts which are logically implied by the formula $\mathcal{P} \wedge \mathcal{A}$. The semantics thus obtained is the *minimal model semantics*. Alternatively, datalog has an operational semantics by viewing the rules of a program as *inference rules* which allow us to derive a fact that appears in the head of a rule if all facts appearing in the body of a rule can be deduced. Nonground clauses are replaced by all possible ground instantiations over the active domain (i.e., the set of domain elements appearing in $\mathcal{P} \wedge \mathcal{A}$). Formally, this operational semantics is defined in terms of the *immediate consequence operator* which augments a given set of facts by those facts which can be inferred in one step by applying the rules in $\mathcal{P}$. The set of (ground) facts obtained as the least fixpoint of the immediate consequence operator coincides with the minimal model of $\mathcal{P} \wedge \mathcal{A}$ (for details see Abiteboul et al. [1995]; Ceri et al. [1990]; Ullman [1989].

Example 2.9. Let us consider the following datalog program $\mathcal{P}$.

$$path(x, y) \leftarrow arc(x, y).$$
$$path(x, y) \leftarrow path(x, z), arc(z, y).$$

We assume the structure $\mathcal{A}$ contains the the following facts: $\{arc(1, 2), arc(2, 3)\}$.

Here the predicate *path* is intensional and *arc* is extensional. Basically, to evaluate the program $\mathcal{P}$ over $\mathcal{A}$, we first conduct the grounding by instantiating the rules over the active domain $\{1, 2, 3\}$. We thus get a ground program with rules $path(1, 2) \leftarrow arc(1, 2)$; $path(2, 1) \leftarrow arc(2, 1)$; $path(1, 3) \leftarrow arc(1, 3)$; ...; $path(1, 2) \leftarrow path(1, 3), arc(3, 2)$; etc. Then, by applying the immediate consequence operator with respect to the grounded rules, we obtain the minimal model of $\mathcal{P} \wedge \mathcal{A}$ where the following facts are true: $\{arc(1, 2), arc(2, 3), path(1, 2), path(2, 3), path(1, 3)\}$.

Concerning the complexity of datalog, we are mainly interested in the combined complexity (i.e., the complexity with respect to the size of the program $\mathcal{P}$ and the size of the data $\mathcal{A}$). In general, the combined complexity of datalog is EXPTIME-complete (implicit in Vardi [1982]). However, there are some fragments which can be evaluated much more efficiently. Below, we give some examples.

(1) *Propositional datalog* (i.e., all rules are ground) can be evaluated in linear time (combined complexity) (see Dowling and Gallier [1984]; Minoux [1988]).

(2) The *guarded fragment* of datalog (i.e., every rule r contains an extensional atom B in the body, such that all variables occurring in r also occur in B) can be evaluated in time $\mathcal{O}(|\mathcal{P}| * |\mathcal{A}|)$ (see Gottlob et al. [2002]). This upper bound on the complexity follows easily from the observation that the "guard" B in a rule r admits at most $|\mathcal{A}|$ possible instantiations that are contained in the extensional database $\mathcal{A}$. Since all variables in r occur in B, also the

number of possible ground instantiations (whose bodies do not contain an extensional atom outside $\mathcal{A}$) of every rule is bounded by $|\mathcal{A}|$. The guarded fragment of first-order logic was introduced in Andréka et al. [1995]. In the context of logic programming, the guarded fragment was first studied in Gottlob et al. [2002] and further treated in Calì et al. [2009a, 2009b].

(3) *Monadic datalog* (i.e., all intensional predicates are unary) is NP-complete (combined complexity) (see Gottlob and Koch [2004]).

(4) In Foustoucos and Guessarian [2006], the tractability of some fragments of *inf-datalog* was shown. Inf-datalog extends the usual least fixpoint semantics of datalog with greatest fixpoint. It thus captures some modal logics that play an important role in computer-aided verification.

3. INDUCED SUBSTRUCTURES

In this section, we study the k-types of substructures induced by certain subtrees of a tree decomposition (see Definitions 3.1 and 3.2). Moreover, it is convenient to introduce some additional notation in Definition 3.4 below.

Definition 3.1. Let T be a tree and t a node in T. Then we denote the *subtree rooted at t* as T_t. Moreover, analogously to Neven and Schwentick [2002], we write $\bar{T}_t$ to denote the *envelope of* T_t. This envelope is obtained by removing all of T_t from T except for the node t.

Likewise, let $\mathcal{T} = \langle T, (A_s)_{s\in T}\rangle$ be a tree decomposition of a structure. Then we define $\mathcal{T}_t = \langle T_t, (A_s)_{s\in T_t}\rangle$ and $\bar{\mathcal{T}}_t = \langle \bar{T}_t, (A_s)_{s\in \bar{T}_t}\rangle$.

In other words, t is the root node in T_t while, in $\bar{T}_t$, it is a leaf node. Clearly, the only node occurring in both T_t and $\bar{T}_t$ is t.

Definition 3.2. Let $\mathcal{A}$ be a structure and let $\mathcal{T} = \langle T, (A_t)_{t\in T}\rangle$ be a tree decomposition of $\mathcal{A}$. Moreover, let s be a node in $\mathcal{T}$ with bag $A_s = \bar{a} = (a_0, \ldots, a_w)$ and let $\mathcal{S}$ be one of the subtrees $\mathcal{T}_s$ or $\bar{\mathcal{T}}_s$ of $\mathcal{T}$.

Then we write $\mathcal{I}(\mathcal{A}, \mathcal{S}, s)$ to denote the structure $(\mathcal{A}', \bar{a})$, where $\mathcal{A}'$ is the substructure of $\mathcal{A}$ induced by the elements occurring in the bags of $\mathcal{S}$.

Example 3.3. Recall the relational schema (R, F) represented by the structure $\mathcal{A}$ from Example 2.2 with (normalized) tree decomposition $\mathcal{T}'$ in Figure 2. Consider, for instance, the node s in $\mathcal{T}'$, as depicted in Figure 4, with bag $A_s = (f_3, b, c)$. Then the induced substructure $\mathcal{I}(\mathcal{A}, \mathcal{T}'_s, s)$ is the substructure of $\mathcal{A}$ which is induced by the elements occurring in the bags of $\mathcal{T}'_s$, whereas $\mathcal{I}(\mathcal{A}, \bar{\mathcal{T}}'_s, s)$ the substructure of $\mathcal{A}$ which is induced by the elements occurring in the bags of $\bar{\mathcal{T}}'_s$.

Definition 3.4. Let $w \geq 1$ be a natural number and let $\mathcal{A}$ and $\mathcal{B}$ be τ-structures for some signature τ. Moreover, let $(a_0, \ldots, a_w)$ (respectively, $(b_0, \ldots, b_w)$) be a tuple of pairwise distinct elements in A (respectively, B).

We call $(a_0, \ldots, a_w)$ and $(b_0, \ldots, b_w)$ *equivalent* and write $(a_0, \ldots, a_w) \equiv (b_0, \ldots, b_w)$ if and only if for any predicate symbol $R \in \tau$ with arity α and for all tuples $(i_1, \ldots, i_\alpha) \in \{0, \ldots, w\}^\alpha$, the equivalence $R^{\mathcal{A}}(a_{i_1}, \ldots, a_{i_\alpha}) \Leftrightarrow R^{\mathcal{B}}(b_{i_1}, \ldots, b_{i_\alpha})$ holds.

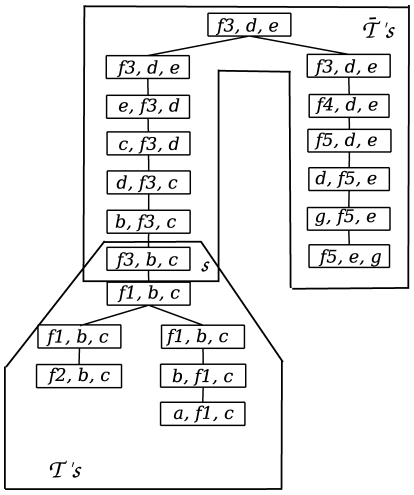

Fig. 4. Induced substructures $\mathcal{T}'_s$ and $\bar{\mathcal{T}}'_s$ of the tree decomposition $\mathcal{T}$ w.r.t. the node s.

We are now ready to generalize results from Neven and Schwentick [2002] (dealing with trees together with a distinguished node which is either the root or some leaf node) to the case of structures of bounded treewidth over an arbitrary signature τ. In the three lemmas below, let $k \geq 0$ and $w \geq 1$ be arbitrary natural numbers and let τ be an arbitrary signature.

LEMMA 3.5. *Let $\mathcal{A}$ and $\mathcal{B}$ be τ-structures, let $\mathcal{S}$ (respectively, $\mathcal{T}$) be a normalized tree decomposition of $\mathcal{A}$ (respectively, of $\mathcal{B}$) of width w, and let s (respectively, t) be an internal node in $\mathcal{S}$ (respectively, in $\mathcal{T}$).*

(1) Permutation nodes. *Let s' (respectively, t') be the only child of s in $\mathcal{S}$ (respectively, of t in $\mathcal{T}$). Moreover, let $\bar{a}$, $\bar{a}'$, $\bar{b}$, and $\bar{b}'$ denote the bags at the nodes s, s', t, and t', respectively.*
If $\mathcal{I}(\mathcal{A}, \mathcal{S}_{s'}, s') \equiv_k^{MSO} \mathcal{I}(\mathcal{B}, \mathcal{T}_{t'}, t')$ and there exists a permutation π, such that $\bar{a} = \pi(\bar{a}')$ and $\bar{b} = \pi(\bar{b}')$, then $\mathcal{I}(\mathcal{A}, \mathcal{S}_s, s) \equiv_k^{MSO} \mathcal{I}(\mathcal{B}, \mathcal{T}_t, t)$.

(2) Element replacement nodes. *Let s' (respectively, t') be the only child of s in $\mathcal{S}$ (respectively, of t in $\mathcal{T}$). Moreover, let $\bar{a} = (a_0, a_1, \ldots, a_w)$, $\bar{a}' = (a'_0, a_1, \ldots, a_w)$, $\bar{b} = (b_0, b_1, \ldots, b_w)$, and $\bar{b}' = (b'_0, b_1, \ldots, b_w)$ denote the bags at the nodes s, s', t, and t', respectively.*
If $\mathcal{I}(\mathcal{A}, \mathcal{S}_{s'}, s') \equiv_k^{MSO} \mathcal{I}(\mathcal{B}, \mathcal{T}_{t'}, t')$ and $\bar{a} \equiv \bar{b}$, then $\mathcal{I}(\mathcal{A}, \mathcal{S}_s, s) \equiv_k^{MSO} \mathcal{I}(\mathcal{B}, \mathcal{T}_t, t)$.

(3) Branch nodes. *Let s_1 and s_2 (respectively, t_1 and t_2) be the children of s in $\mathcal{S}$ (respectively, of t in $\mathcal{T}$).*
If $\mathcal{I}(\mathcal{A}, \mathcal{S}_{s_1}, s_1) \equiv_k^{MSO} \mathcal{I}(\mathcal{B}, \mathcal{T}_{t_1}, t_1)$ and $\mathcal{I}(\mathcal{A}, \mathcal{S}_{s_2}, s_2) \equiv_k^{MSO} \mathcal{I}(\mathcal{B}, \mathcal{T}_{t_2}, t_2)$, then $\mathcal{I}(\mathcal{A}, \mathcal{S}_s, s) \equiv_k^{MSO} \mathcal{I}(\mathcal{B}, \mathcal{T}_t, t)$.

LEMMA 3.6. *Let $\mathcal{A}$ and $\mathcal{B}$ be τ-structures, let $\mathcal{S}$ (respectively, $\mathcal{T}$) be a normalized tree decomposition of $\mathcal{A}$ (respectively, of $\mathcal{B}$) of width w, and let s (respectively, t) be an internal node in $\mathcal{S}$ (respectively, in $\mathcal{T}$).*

(1) Permutation nodes. *Let s' (respectively, t') be the only child of s in $\mathcal{S}$ (respectively, of t in $\mathcal{T}$). Moreover, let $\bar{a}$, $\bar{a}'$, $\bar{b}$, and $\bar{b}'$ denote the bags at the nodes s, s', t, and t', respectively.*

If $\mathcal{I}(\mathcal{A}, \bar{\mathcal{S}}_s, s) \equiv_k^{MSO} \mathcal{I}(\mathcal{B}, \bar{\mathcal{T}}_t, t)$ *and there exists a permutation* π*, such that* $\bar{a} = \pi(\bar{a}')$ *and* $\bar{b} = \pi(\bar{b}')$*, then* $\mathcal{I}(\mathcal{A}, \bar{\mathcal{S}}_{s'}, s') \equiv_k^{MSO} \mathcal{I}(\mathcal{B}, \bar{\mathcal{T}}_{t'}, t')$.

(2) Element replacement nodes. *Let* s' *(respectively,* t'*) be the only child of* s *in* $\mathcal{S}$ *(respectively, of* t *in* $\mathcal{T}$*). Moreover, let* $\bar{a} = (a_0, a_1, \ldots, a_w)$*,* $\bar{a}' = (a_0', a_1, \ldots, a_w)$*,* $\bar{b} = (b_0, b_1, \ldots, b_w)$*, and* $\bar{b}' = (b_0', b_1, \ldots, b_w)$ *denote the bags at the nodes* s*,* s'*,* t*, and* t'*, respectively.*

If $\mathcal{I}(\mathcal{A}, \bar{\mathcal{S}}_s, s) \equiv_k^{MSO} \mathcal{I}(\mathcal{B}, \bar{\mathcal{T}}_t, t)$ *and* $\bar{a}' \equiv \bar{b}'$*, then* $\mathcal{I}(\mathcal{A}, \bar{\mathcal{S}}_{s'}, s') \equiv_k^{MSO} \mathcal{I}(\mathcal{B}, \bar{\mathcal{T}}_{t'}, t')$.

(3) Branch nodes. *Let* s_1 *and* s_2 *(respectively,* t_1 *and* t_2*) be the children of* s *in* $\mathcal{S}$ *(respectively, of* t *in* $\mathcal{T}$*). If* $\mathcal{I}(\mathcal{A}, \bar{\mathcal{S}}_s, s) \equiv_k^{MSO} \mathcal{I}(\mathcal{B}, \bar{\mathcal{T}}_t, t)$ *and* $\mathcal{I}(\mathcal{A}, \mathcal{S}_{s_2}, s_2) \equiv_k^{MSO} \mathcal{I}(\mathcal{B}, \mathcal{T}_{t_2}, t_2)$*, then* $\mathcal{I}(\mathcal{A}, \bar{\mathcal{S}}_{s_1}, s_1) \equiv_k^{MSO} \mathcal{I}(\mathcal{B}, \bar{\mathcal{T}}_{t_1}, t_1)$.

If $\mathcal{I}(\mathcal{A}, \bar{\mathcal{S}}_s, s) \equiv_k^{MSO} \mathcal{I}(\mathcal{B}, \bar{\mathcal{T}}_t, t)$ *and* $\mathcal{I}(\mathcal{A}, \mathcal{S}_{s_1}, s_1) \equiv_k^{MSO} \mathcal{I}(\mathcal{B}, \mathcal{T}_{t_1}, t_1)$*, then* $\mathcal{I}(\mathcal{A}, \bar{\mathcal{S}}_{s_2}, s_2) \equiv_k^{MSO} \mathcal{I}(\mathcal{B}, \bar{\mathcal{T}}_{t_2}, t_2)$.

LEMMA 3.7. *Let* $\mathcal{A}$ *and* $\mathcal{B}$ *be* τ*-structures, let* $\mathcal{S}$ *(respectively,* $\mathcal{T}$*) be a normalized tree decomposition of* $\mathcal{A}$ *(respectively, of* $\mathcal{B}$*) of width* w*, and let* s *(respectively,* t*) be an arbitrary node in* $\mathcal{S}$ *(respectively, in* $\mathcal{T}$*), whose bag is* $(a_0, \ldots, a_w)$ *(respectively,* $(b_0, \ldots, b_w)$*).*
If $\mathcal{I}(\mathcal{A}, \mathcal{S}_s, s) \equiv_k^{MSO} \mathcal{I}(\mathcal{B}, \mathcal{T}_t, t)$ *and* $\mathcal{I}(\mathcal{A}, \bar{\mathcal{S}}_s, s) \equiv_k^{MSO} \mathcal{I}(\mathcal{B}, \bar{\mathcal{T}}_t, t)$*, then* $(\mathcal{A}, a_i) \equiv_k^{MSO} (\mathcal{B}, b_i)$ *for every* $i \in \{0, \ldots, w\}$.

PROOF IDEA OF THE LEMMAS. The three lemmas can be proved by Ehrenfeucht-Fraïssé games (see Ebbinghaus and Flum [1999]; Libkin [2004]). In all cases, we extend or combine the winning strategy of the duplicator on the original pair(s) of structures to a winning strategy on the target structures. Clearly, the elements selected by the two players in point moves define a partial isomorphism on the original pair(s) of substructures. The connectedness condition of tree decompositions allows us to conclude that these elements also define a partial isomorphism on the target substructures. Alternatively, these lemmas can be inferred from the proof of Courcelle's Theorem in Courcelle [1990b].

Remark. The three lemmas follow the spirit of *composition theorems* like the Feferman Vaught Theorem (see Makowsky [2004] for various forms and applications of this famous theorem). Lemma 3.5 states that the k-type of the substructure induced by a subtree $\mathcal{S}_s$ of the tree decomposition $\mathcal{S}$ is fully determined by the type of the structure induced by the subtree rooted at the child node(s) together with the relations between elements in the bag at node s. Analogously, Lemma 3.6 deals with the k-type of the substructure induced by a subtree $\bar{\mathcal{S}}_s$. Finally, Lemma 3.7 states that the k-type of the substructures induced by $\mathcal{S}_s$ and $\bar{\mathcal{S}}_s$ fully determines the type of the entire structure $\mathcal{A}$ extended by some domain element from the bag of s.

4. MONADIC DATALOG

In this section, we introduce two restricted fragments of datalog, namely, *monadic datalog* over structures of bounded treewidth and the *quasiguarded fragment* of datalog. Let $\tau = \{R_1, \ldots, R_K\}$ be a set of predicate symbols and let $w \geq 1$ denote the treewidth. We define the following extended signature τ_{td}.

$$\tau_{td} = \tau \cup \{root, leaf, child_1, child_2, bag\},$$

where the unary predicates *root*, and *leaf* as well as the binary predicates $child_1$ and $child_2$ are used to represent the tree T of the normalized tree decomposition in the obvious way. For instance, we write $child_1(s_1, s)$ to denote that s_1 is either the first child or the only child of s. Finally, *bag* has arity $w + 2$, where $bag(t, a_0, \ldots, a_w)$ means that the bag at node t is $(a_0, \ldots, a_w)$.

Definition 4.1. Let τ be a set of predicate symbols and let $w \geq 1$. A monadic datalog program over τ-structures of treewidth w is a set of datalog rules where all extensional predicates are from τ_{td} and all intensional predicates are unary.

Let $\mathcal{A}$ be a τ-structure $\mathcal{A}$ and let $\mathcal{T} = \langle T, (A_t)_{t \in T} \rangle$ be an arbitrary, normalized tree decomposition of $\mathcal{A}$ of width w. Then we denote by $\mathcal{A}_{td}$ the τ_{td}-structure representing $\mathcal{A}$ and $\mathcal{T}$ as follows: the domain of $\mathcal{A}_{td}$ is the union of $dom(\mathcal{A})$ and the set of nodes of T. In addition to the relations $R_i^{\mathcal{A}}$ with $R_i \in \tau$, the structure $\mathcal{A}_{td}$ also contains relations for each predicate *root*, *leaf*, $child_1$, $child_2$, and *bag* thus representing the tree decomposition $\mathcal{T}$. In the sequel, we shall refer to $\mathcal{A}_{td}$ as the *decomposed structure* or as a *structure decomposing* $\mathcal{A}$. By Bodlaender [1996], one can compute $\mathcal{T}$ from $\mathcal{A}$ in linear time with respect to the size of $\mathcal{A}$. Hence, the size of $\mathcal{A}_{td}$ (for some reasonable encoding, e.g., the one presented in Flum et al. [2002], which we recalled in Section 2.2) is also linearly bounded by the size of $\mathcal{A}$.

Example 4.2. Recall the relational schema (R, F) represented by the structure $\mathcal{A}$ from Example 2.2 with normalized tree decomposition $\mathcal{T}'$ in Figure 2. The domain of $\mathcal{A}_{td}$ is the union of $dom(\mathcal{A})$ and the set of nodes $\{s_1, \ldots, s_{22}\}$. The corresponding τ_{td}-structure $\mathcal{A}_{td}$ representing the relational schema together with tree decomposition $\mathcal{T}'$ is made up by the following set of ground atoms: $root(s_1)$, $leaf(s_{12})$, $leaf(s_{14})$, $leaf(s_{19})$, $child_1(s_2, s_1)$, $child_2(s_3, s_1)$, $\ldots$, $bag(s_1, f_3, d, e)$, $\ldots$.

As we recalled in Section 2.4, the evaluation of monadic datalog is NP-complete (combined complexity). However, the target of our transformation from MSO to datalog will be a further restricted fragment of datalog, which we refer to as *quasiguarded*. The evaluation of this fragment is tractable.

Definition 4.3. Let τ be a finite set of predicate symbols and let $\mathcal{P}$ be a datalog program over the extended signature τ_{td} for some treewidth $w \geq 1$. Moreover, let r be a rule in $\mathcal{P}$ and let x, y be variables in r.

We say that y is *functionally dependent* on x in one step, if the body of r contains an atom of one of the following forms: $child_1(x, y)$, $child_1(y, x)$, $child_2(x, y)$, $child_2(y, x)$, or $bag(x, a_0, \ldots, a_k)$ with $y = a_i$ for some $i \in \{1, \ldots, k\}$.

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

Definition 4.4. Let τ be a finite set of predicate symbols and let $\mathcal{P}$ be a datalog program over the extended signature τ_{td} for some treewidth $w \geq 1$. We call a datalog program $\mathcal{P}$ over τ_{td} *quasiguarded* if every rule r in $\mathcal{P}$ contains an extensional atom B, such that every variable occurring in r either occurs in B

or is functionally dependent on some variable in B. If this is the case, we call B a *quasiguard* of r.

The idea of the above definitions is the following: suppose that we have a τ_{td}-structure $\mathcal{A}$ and a ground instantiation of some rule r, such that each extensional atom in r is indeed instantiated to some atom in $\mathcal{A}$. If a variable y is functionally dependent on x then the value of y is fully determined by x, that is, for every ground instantiation of x there exists at most one ground instantiation of y. Hence, in a quasiguarded datalog program, every rule r has at most $|\mathcal{A}|$ possible ground instantiations such that the extensional atoms of r are instantiated to atoms in $\mathcal{A}$—this corresponds to the maximally $|\mathcal{A}|$ ground instantiations of the quasiguard in the rule r.

THEOREM 4.5. *Let $\mathcal{P}$ be a quasiguarded datalog program and let $\mathcal{A}$ be a structure. Then $\mathcal{P}$ can be evaluated over $\mathcal{A}$ in time $\mathcal{O}(|\mathcal{P}| * |\mathcal{A}|)$, where $|\mathcal{P}|$ denotes the size of the datalog program and $|\mathcal{A}|$ denotes the size of the data.*

PROOF. Let r be a rule in the program $\mathcal{P}$ and let B be the "quasiguard" of r, that is, all variables in r either occur in B or are functionally dependent on some variable in B. Clearly, the semantics of $\mathcal{P} \cup \mathcal{A}$ is not changed if we replace each rule r in $\mathcal{P}$ by the set of all possible ground instances r' of r, such that the extensional atoms in r' are indeed contained in $\mathcal{A}$. In order to compute all these ground instances r', we first instantiate B. The maximal number of such instantiations is bounded by $|\mathcal{A}|$.

Now consider the remaining variables in r. By assumption, they are all functionally dependent on the variables in B. Hence, we can iteratively determine the only possible value of the variables which are functionally dependent on some variable in B in i steps (for $i \geq 1$). By the above considerations, all variables outside B admit at most one ground instantiation such that the extensional atoms of r are contained in $\mathcal{A}$. Hence, the number of all possible ground instantiations r' of r is bounded by $|\mathcal{A}|$ and all these ground instantiations can be computed in time $\mathcal{O}(|\mathcal{A}|)$.

Hence, in total, the ground program $\mathcal{P}'$ consisting of all possible ground instantiations of the rules in $\mathcal{P}$ has size $\mathcal{O}(|\mathcal{P}| * |\mathcal{A}|)$ and also the computation of these ground rules fits into the linear time bound. As we recalled in Section 2.4, the ground program $\mathcal{P}'$ can be evaluated over $\mathcal{A}$ in time $\mathcal{O}(|\mathcal{P}'| + |\mathcal{A}|) = \mathcal{O}((|\mathcal{P}| * |\mathcal{A}|) + |\mathcal{A}|) = \mathcal{O}(|\mathcal{P}| * |\mathcal{A}|)$. □

Before we state the main result concerning the *expressive power* of monadic datalog over structures of bounded treewidth, we introduce the following notation. In order to simplify the exposition below, we assume that all predicates $R_i \in \tau$ have the same arity r. First, this can be easily achieved by copying columns in relations with smaller arity. Moreover, it is easily seen that the results also hold without this restriction.

It is convenient to use the following abbreviations. Let $\bar{a} = (a_0, \ldots, a_w)$ be a tuple of domain elements. Then we write $\mathcal{R}(\bar{a})$ to denote the set of all ground atoms with predicates in $\tau = \{R_1, \ldots, R_K\}$ and arguments in $\{a_0, \ldots, a_w\}$,

that is,

$$\mathcal{R}(\bar{a}) = \bigcup_{i=1}^{K} \bigcup_{j_1=0}^{w} \dots \bigcup_{j_r=0}^{w} \{R_i(a_{j_1}, \dots, a_{j_r})\}.$$

Let $\mathcal{A}$ be a structure with tree decomposition $\mathcal{T}$ and let s be a node in $\mathcal{T}$ whose bag is $\bar{a} = (a_0, \dots, a_w)$. Then we write $(\mathcal{A}, s)$ as a short-hand for the structure $(\mathcal{A}, \bar{a})$ with distinguished constants $\bar{a} = (a_0, \dots, a_w)$.

THEOREM 4.6. *Let τ and $w \geq 1$ be arbitrary but fixed. Every MSO-definable unary query over τ-structures of treewidth w is also computable by a quasi-guarded monadic datalog program over τ_{td}.*

PROOF. Let $\varphi(x)$ be an arbitrary MSO formula with free variable x and quantifier depth k. We have to construct a monadic datalog program $\mathcal{P}$ with distinguished predicate φ which defines the same query.

Without loss of generality, we only consider the case of structures whose domain has at least $w+1$ elements. We maintain two disjoint sets of tokens $\Theta^{\uparrow}$ and $\Theta^{\downarrow}$, representing k-types of structures $(\mathcal{A}, \bar{a})$ of the following form: $\mathcal{A}$ has a tree decomposition $\mathcal{T}$ of width w and $\bar{a}$ is the bag of some node s in $\mathcal{T}$. Moreover, for $\Theta^{\uparrow}$, we require that s be the root of $\mathcal{S}$ while, for $\Theta^{\downarrow}$, we require that s be a leaf node of $\mathcal{T}$. In order to ensure that $\Theta^{\uparrow}$ and $\Theta^{\downarrow}$ are indeed disjoint, every token in $\Theta^{\uparrow}$ (respectively, in $\Theta^{\downarrow}$) is of the form $\vartheta\uparrow$ (respectively, $\vartheta\downarrow$) representing some type ϑ. We maintain for each token $\vartheta\uparrow$ (respectively, $\vartheta\downarrow$) representing a type ϑ a witness $W(\vartheta\uparrow) = \langle \mathcal{A}, \mathcal{T}, s\rangle$ (respectively, $W(\vartheta\downarrow) = \langle \mathcal{A}, \mathcal{T}, s\rangle$). The tokens in $\Theta^{\uparrow}$ and $\Theta^{\downarrow}$ will serve as predicate names in the monadic datalog program to be constructed. Initially, $\Theta^{\uparrow} = \Theta^{\downarrow} = \mathcal{P} = \emptyset$.

Below we describe a bottom-up construction of $\Theta^{\uparrow}$ and a top-down construction of $\Theta^{\downarrow}$, respectively. Note that these constructions do not refer to a particular term or structure. Instead, the goal of these constructions is to generate tokens representing the types of *all* possible structures $(\mathcal{A}, \bar{a})$ of the above mentioned form.

(1) *"Bottom-up" construction of* $\Theta^{\uparrow}$.

BASE CASE. In this step, we construct all possible types ϑ of structures $(\mathcal{A}, \bar{a})$ whose tree decomposition consists of a single node. Let $a_0, \dots, a_w$ be pairwise distinct objects and let $\mathcal{S}$ be a tree decomposition consisting of a single node s, whose bag is $A_s = (a_0, \dots, a_w)$. Then we consider all possible structures $(\mathcal{A}, s)$ with this tree decomposition. In particular, $dom(\mathcal{A}) = \{a_0, \dots, a_w\}$. We get all possible structures with tree decomposition $\mathcal{S}$ by letting the EDB $\mathcal{E}(\mathcal{A})$ be any subset of $\mathcal{R}(\bar{a})$. For every such structure $(\mathcal{A}, s)$, we check if there exists a token $\vartheta\uparrow \in \Theta^{\uparrow}$ with $W(\vartheta\uparrow) = \langle \mathcal{B}, \mathcal{T}, t\rangle$, such that $(\mathcal{A}, s) \equiv_k^{MSO} (\mathcal{B}, t)$. If such a $\vartheta\uparrow$ exists, we take it. Otherwise we invent a new token $\vartheta\uparrow$, add it to $\Theta^{\uparrow}$, and set $W(\vartheta\uparrow) := \langle \mathcal{A}, \mathcal{S}, s\rangle$. In any case, we add the following rule to the program $\mathcal{P}$:

$$\vartheta\uparrow(v) \leftarrow bag(v, x_0, \dots, x_w), leaf(v), \{R_i(x_{j_1}, \dots, x_{j_r}) \mid R_i(a_{j_1}, \dots, a_{j_r}) \in \mathcal{E}(\mathcal{A})\}, \{\neg R_i(x_{j_1}, \dots, x_{j_r}) \mid R_i(a_{j_1}, \dots, a_{j_r}) \notin \mathcal{E}(\mathcal{A})\}.$$

INDUCTION STEP. We construct new structures by extending the tree decompositions of existing witnesses in "bottom-up" direction, that is, by

introducing a new root node. This root node may be one of three kinds of nodes.

(a) *Permutation nodes.* For each $\vartheta\uparrow' \in \Theta^{\uparrow}$, let $W(\vartheta\uparrow') = \langle \mathcal{A}, \mathcal{S}', s' \rangle$ with bag $A_{s'} = (a_0, \ldots, a_w)$ at the root s' in $\mathcal{S}'$. Then we consider all possible triples $\langle \mathcal{A}, \mathcal{S}, s \rangle$, where $\mathcal{S}$ is obtained from $\mathcal{S}'$ by appending s' to a new root node s, such that s is a permutation node, that is, there exists some permutation π, such that $A_s = (a_{\pi(0)}, \ldots, a_{\pi(w)})$

For every such structure $(\mathcal{A}, s)$, we check if there exists a token $\vartheta\uparrow \in \Theta^{\uparrow}$ with $W(\vartheta\uparrow) = \langle \mathcal{B}, \mathcal{T}, t \rangle$, such that $(\mathcal{A}, s) \equiv_k^{MSO} (\mathcal{B}, t)$. If such a $\vartheta\uparrow$ exists, we take it. Otherwise we invent a new token $\vartheta\uparrow$, add it to $\Theta^{\uparrow}$, and set $W(\vartheta\uparrow) := \langle \mathcal{A}, \mathcal{S}, s \rangle$. In any case, we add the following rule to the program $\mathcal{P}$:

$$\vartheta\uparrow(v) \leftarrow \mathit{bag}(v, x_{\pi(0)}, \ldots, x_{\pi(w)}), \mathit{child}_1(v', v), \vartheta\uparrow'(v'), \mathit{bag}(v', x_0, \ldots, x_w).$$

(b) *Element replacement nodes.* For each $\vartheta\uparrow' \in \Theta^{\uparrow}$, let $W(\vartheta\uparrow') = \langle \mathcal{A}', \mathcal{S}', s' \rangle$ with bag $A_{s'} = (a'_0, a_1, \ldots, a_w)$ at the root s' in $\mathcal{S}'$. Then we consider all possible triples $\langle \mathcal{A}, \mathcal{S}, s \rangle$, where $\mathcal{S}$ is obtained from $\mathcal{S}'$ by appending s' to a new root node s, such that s is an element replacement node. For the tree decomposition $\mathcal{S}$, we thus invent some new element a_0 and set $A_s = (a_0, a_1, \ldots, a_w)$. For this tree decomposition $\mathcal{S}$, we consider all possible structures $\mathcal{A}$ with $\mathit{dom}(\mathcal{A}) = \mathit{dom}(A') \cup \{a_0\}$, where the EDB $\mathcal{E}(\mathcal{A}')$ is extended to the EDB $\mathcal{E}(\mathcal{A})$ by new ground atoms from $\mathcal{R}(\bar{a})$, such that a_0 occurs as argument of all ground atoms in $\mathcal{E}(\mathcal{A}) \setminus \mathcal{E}(\mathcal{A}')$.

For every such structure $(\mathcal{A}, s)$, we check if there exists a token $\vartheta\uparrow \in \Theta^{\uparrow}$ with $W(\vartheta\uparrow) = \langle \mathcal{B}, \mathcal{T}, t \rangle$, such that $(\mathcal{A}, s) \equiv_k^{MSO} (\mathcal{B}, t)$. If such a $\vartheta\uparrow$ exists, we take it. Otherwise we invent a new token $\vartheta\uparrow$, add it to $\Theta^{\uparrow}$, and set $W(\vartheta\uparrow) := \langle \mathcal{A}, \mathcal{S}, s \rangle$. In any case, we add the following rule to the program $\mathcal{P}$:

$$\begin{aligned}\vartheta\uparrow(v) \leftarrow\ & \mathit{bag}(v, x_0, x_1, \ldots, x_w), \mathit{child}_1(v', v), \vartheta\uparrow'(v'), \mathit{bag}(v', x'_0, x_1, \ldots, x_w), \\ & \{R_i(x_{j_1}, \ldots, x_{j_r}) \mid R_i(a_{j_1}, \ldots, a_{j_r}) \in \mathcal{E}(\mathcal{A})\}, \\ & \{\neg R_i(x_{j_1}, \ldots, x_{j_r}) \mid R_i(a_{j_1}, \ldots, a_{j_r}) \notin \mathcal{E}(\mathcal{A})\}.\end{aligned}$$

(c) *Branch nodes.* Let $\vartheta\uparrow_1$, $\vartheta\uparrow_2$ be two (not necessarily distinct) tokens in $\Theta^{\uparrow}$ with $W(\vartheta\uparrow_1) = \langle \mathcal{A}_1, \mathcal{S}_1, s_1 \rangle$ and $W(\vartheta\uparrow_2) = \langle \mathcal{A}_2, \mathcal{S}_2, s_2 \rangle$. Let $A_{s_1} = (a_0, \ldots, a_w)$ and $A_{s_2} = (b_0, \ldots, b_w)$, respectively. We can assume that $\mathit{dom}(\mathcal{A}_1) \cap \mathit{dom}(\mathcal{A}_2) = \emptyset$ because we can replace every witness by a variable disjoint (isomorphic) copy of it.

Let δ be a renaming function with $\delta = \{a_0 \leftarrow b_0, \ldots, a_w \leftarrow b_w\}$. By applying δ to $\langle \mathcal{A}_2, \mathcal{S}_2, s_2 \rangle$, we obtain a new triple $\langle \mathcal{A}'_2, \mathcal{S}'_2, s_2 \rangle$ with $\mathcal{A}'_2 = \mathcal{A}_2\delta$ and $\mathcal{S}'_2 = \mathcal{S}_2\delta$. In particular, we thus have $A_{s_2}\delta = (a_0, \ldots, a_w)$. Clearly, $(\mathcal{A}_2, s_2) \equiv_k^{MSO} (\mathcal{A}'_2, s_2)$ holds.

For every such pair $\langle \mathcal{A}_1, \mathcal{S}_1, s_1 \rangle$ and $\langle \mathcal{A}'_2, \mathcal{S}'_2, s_2 \rangle$, we check if the EDBs are inconsistent, that is, $\mathcal{E}(\mathcal{A}_1) \cap \mathcal{R}(\bar{a}) \neq \mathcal{E}(\mathcal{A}'_2) \cap \mathcal{R}(\bar{a})$. If this is the case, then we ignore this pair. Otherwise, we construct a new tree decomposition $\mathcal{S}$ with a new root node s, whose child nodes are s_1 and s_2. As the bag of s, we set $A_s = A_{s_1} = A_{s_2}\delta$. By construction, $\mathcal{S}$ is a normalized tree decomposition of the structure $\mathcal{A}$ with $\mathit{dom}(\mathcal{A}) = \mathit{dom}(\mathcal{A}_1) \cup \mathit{dom}(\mathcal{A}'_2)$ and EDB $\mathcal{E}(\mathcal{A}) = \mathcal{E}(\mathcal{A}_1) \cup \mathcal{E}(\mathcal{A}'_2)$.

As in the cases above, we have to check if there exists a token $\vartheta\uparrow \in \Theta^{\uparrow}$ with $W(\vartheta\uparrow) = \langle \mathcal{B}, \mathcal{T}, t\rangle$, such that $(\mathcal{A}, s) \equiv_k^{MSO} (\mathcal{B}, t)$. If such a $\vartheta\uparrow$ exists, we take it. Otherwise we invent a new token $\vartheta\uparrow$, add it to $\Theta^{\uparrow}$, and set $W(\vartheta\uparrow) := \langle \mathcal{A}, \mathcal{S}, s\rangle$. In any case, we add the following rule to the program $\mathcal{P}$:

$$\vartheta\uparrow(v) \leftarrow bag(v, x_0, x_1, \ldots, x_w), child_1(v_1, v), \vartheta\uparrow_1(v_1), child_2(v_2, v), \vartheta\uparrow_2(v_2), bag(v_1, x_0, x_1, \ldots, x_w), bag(v_2, x_0, x_1, \ldots, x_w).$$

As a result of this bottom-up construction, the set $\Theta^{\uparrow}$ is a set of tokens $\vartheta\uparrow$ representing all possible types ϑ of structures $(\mathcal{B}, t)$, where t is the root of a tree decomposition $\mathcal{T}$ of $\mathcal{B}$. We shall come back to this point at the end of the construction of the desired datalog program.

(2) *"Top-down" construction of* $\Theta^{\downarrow}$. Analogously to the "bottom-up" construction of $\Theta^{\uparrow}$, we construct the set $\Theta^{\downarrow}$ with a "top-down" intuition. The base case is essentially the same as before since, in every tree decomposition with only one node s, this single node is both the root and a leaf. For the induction step, we have to select the witness $W(\vartheta\downarrow') = \langle \mathcal{A}', \mathcal{S}', s'\rangle$ of some already computed token $\vartheta\downarrow' \in \Theta^{\downarrow}$. Now the node s' in $\mathcal{S}'$ is a leaf node and we extend $\mathcal{S}'$ to a new tree decomposition $\mathcal{S}$ by appending a new leaf node s as a child of s'. For all such tree decompositions $\mathcal{S}$, we consider all possible structures $\mathcal{A}$ by appropriately extending $\mathcal{A}'$. The rules added to the program $\mathcal{P}$ again reflect the type transitions from the type of the original structure $(\mathcal{A}', s')$ to the type of any such new structure $(\mathcal{A}, s)$.

BASE CASE. Let $a_0, \ldots, a_w$ be pairwise distinct elements and let $\mathcal{S}$ be a tree decomposition consisting of a single node s, whose bag is $A_s = (a_0, \ldots, a_w)$. Then we consider all possible structures $(\mathcal{A}, s)$ with this tree decomposition. In particular, $dom(\mathcal{A}) = \{a_0, \ldots, a_w\}$. We get all possible structures with tree decomposition $\mathcal{S}$ by letting the EDB $\mathcal{E}(\mathcal{A})$ be any subset of $\mathcal{R}(\bar{a})$. For every such structure $(\mathcal{A}, s)$, we check if there exists a token $\vartheta\downarrow \in \Theta^{\downarrow}$ with $W(\vartheta\downarrow) = \langle \mathcal{B}, \mathcal{T}, t\rangle$, such that $(\mathcal{A}, s) \equiv_k^{MSO} (\mathcal{B}, t)$. If such a $\vartheta\downarrow$ exists, we take it. Otherwise we invent a new token $\vartheta\downarrow$, add it to $\Theta^{\downarrow}$, and set $W(\vartheta\downarrow) := \langle \mathcal{A}, \mathcal{S}, s\rangle$. In any case, we add the following rule to the program $\mathcal{P}$:

$$\vartheta\downarrow(v) \leftarrow bag(v, x_0, \ldots, x_w), root(v), \{R_i(x_{j_1}, \ldots, x_{j_r}) \mid R_i(a_{j_1}, \ldots, a_{j_r}) \in \mathcal{E}(\mathcal{A})\}, \{\neg R_i(x_{j_1}, \ldots, x_{j_r}) \mid R_i(a_{j_1}, \ldots, a_{j_r}) \notin \mathcal{E}(\mathcal{A})\}.$$

INDUCTION STEP. We construct new structures by extending the tree decompositions of existing witnesses in "top-down" direction, that is, by introducing a new leaf node s and appending it as new child to a former leaf node s'. The node s' may thus become one of three kinds of nodes in a normalized tree decomposition.

(a) *Permutation nodes.* For each $\vartheta\downarrow' \in \Theta^{\downarrow}$, let $W(\vartheta\downarrow') = \langle \mathcal{A}, \mathcal{S}', s'\rangle$ with bag $A_{s'} = (a_0, \ldots, a_w)$ at some leaf node s' in $\mathcal{S}'$. Then we consider all possible triples $\langle \mathcal{A}, \mathcal{S}, s\rangle$, where $\mathcal{S}$ is obtained from $\mathcal{S}'$ by appending s as a new child of s', such that s' is a permutation node, that is, there exists some permutation π, such that $A_s = (a_{\pi(0)}, \ldots, a_{\pi(w)})$

For every such structure $(\mathcal{A}, s)$, we check if there exists a token $\vartheta\downarrow \in \Theta^{\downarrow}$ with $W(\vartheta\downarrow) = \langle \mathcal{B}, \mathcal{T}, t\rangle$, such that $(\mathcal{A}, s) \equiv_k^{MSO} (\mathcal{B}, t)$. If such a $\vartheta\downarrow$ exists, we take it. Otherwise we invent a new token $\vartheta\downarrow$, add it to $\Theta^{\downarrow}$, and set $W(\vartheta\downarrow) := \langle \mathcal{A}, \mathcal{S}, s\rangle$. In any case, we add the following rule to the program $\mathcal{P}$:

$$\vartheta\downarrow(v) \leftarrow bag(v, x_{\pi(0)}, \ldots, x_{\pi(w)}), child_1(v, v'), \vartheta\downarrow'(v'), bag(v', x_0, \ldots, x_w).$$

(b) *Element replacement nodes*. For each $\vartheta\downarrow' \in \Theta^{\downarrow}$, let $W(\vartheta\downarrow') = \langle \mathcal{A}', \mathcal{S}', s'\rangle$ with bag $A_{s'} = (a_0', a_1, \ldots, a_w)$ at leaf node s' in $\mathcal{S}'$. Then we consider all possible triples $\langle \mathcal{A}, \mathcal{S}, s\rangle$, where $\mathcal{S}$ is obtained from $\mathcal{S}'$ by appending s as new child of s', such that s' is an element replacement node. For the tree decomposition $\mathcal{S}$, we thus invent some new element a_0 and set $A_s = (a_0, a_1, \ldots, a_w)$. For this tree decomposition $\mathcal{S}$, we consider all possible structures $\mathcal{A}$ with $dom(\mathcal{A}) = dom(\mathcal{A}') \cup \{a_0\}$ where the EDB $\mathcal{E}(\mathcal{A}')$ is extended to the EDB $\mathcal{E}(\mathcal{A})$ by new ground atoms from $\mathcal{R}(\bar{a})$, such that a_0 occurs as argument of all ground atoms in $\mathcal{E}(\mathcal{A}) \setminus \mathcal{E}(\mathcal{A}')$.

For every such structure $(\mathcal{A}, s)$, we check if there exists a token $\vartheta\downarrow \in \Theta^{\downarrow}$ with $W(\vartheta\downarrow) = \langle \mathcal{B}, \mathcal{T}, t\rangle$, such that $(\mathcal{A}, s) \equiv_k^{MSO} (\mathcal{B}, t)$. If such a $\vartheta\downarrow$ exists, we take it. Otherwise we invent a new token $\vartheta\downarrow$, add it to $\Theta^{\downarrow}$, and set $W(\vartheta\downarrow) := \langle \mathcal{A}, \mathcal{S}, s\rangle$. In any case, we add the following rule to the program $\mathcal{P}$:

$$\begin{aligned}\vartheta\downarrow(v) \leftarrow\ & bag(v, x_0, x_1, \ldots, x_w), child_1(v, v'), \vartheta\downarrow'(v'), bag(v', x_0', x_1, \ldots, x_w),\\ & \{R_i(x_{j_1}, \ldots, x_{j_r}) \mid R_i(a_{j_1}, \ldots, a_{j_r}) \in \mathcal{E}(\mathcal{A})\},\\ & \{\neg R_i(x_{j_1}, \ldots, x_{j_r}) \mid R_i(a_{j_1}, \ldots, a_{j_r}) \notin \mathcal{E}(\mathcal{A})\}.\end{aligned}$$

(c) *Branch nodes*. Let $\vartheta\downarrow \in \Theta^{\downarrow}$ and $\vartheta\uparrow_2 \in \Theta^{\uparrow}$ with $W(\vartheta\downarrow) = \langle \mathcal{A}, \mathcal{S}, s\rangle$ and $W(\vartheta\uparrow_2) = \langle \mathcal{A}_2, \mathcal{S}_2, s_2\rangle$. Note that s is a leaf in $\mathcal{S}$ while s_2 is the root of $\mathcal{S}_2$. Now let $A_s = (a_0, \ldots, a_w)$ and $A_{s_2} = (b_0, \ldots, b_w)$, respectively. We can assume that $dom(\mathcal{A}) \cap dom(\mathcal{A}_2) = \emptyset$ because we can replace every witness by a variable disjoint (isomorphic) copy of it.

Let δ be a renaming function with $\delta = \{a_0 \leftarrow b_0, \ldots, a_w \leftarrow b_w\}$. By applying δ to $\langle \mathcal{A}_2, \mathcal{S}_2, s_2\rangle$, we obtain a new triple $\langle \mathcal{A}_2', \mathcal{S}_2', s_2\rangle$ with $\mathcal{A}_2' = \mathcal{A}_2\delta$ and $\mathcal{S}_2' = \mathcal{S}_2\delta$. In particular, we thus have $A_{s_2}\delta = (a_0, \ldots, a_w)$. Clearly, $(\mathcal{A}_2, s_2) \equiv_k^{MSO} (\mathcal{A}_2', s_2)$ holds.

For every such pair $\langle \mathcal{A}, \mathcal{S}, s\rangle$ and $\langle \mathcal{A}_2', \mathcal{S}_2', s_2\rangle$, we check if the EDBs are inconsistent, that is, $\mathcal{E}(\mathcal{A}) \cap \mathcal{R}(\bar{a}) \neq \mathcal{E}(\mathcal{A}_2') \cap \mathcal{R}(\bar{a})$. If this is the case, then we ignore this pair. Otherwise, we construct a new tree decomposition $\mathcal{S}_1$ by introducing a new leaf node s_1 and appending both s_1 and s_2 as child nodes of s. As the bag of s_1, we set $A_{s_1} = A_s = A_{s_2}\delta$. By construction, $\mathcal{S}_1$ is a normalized tree decomposition of the structure $\mathcal{A}_1$ with $dom(\mathcal{A}_1) = dom(\mathcal{A}) \cup dom(\mathcal{A}_2')$ and EDB $\mathcal{E}(\mathcal{A}_1) = \mathcal{E}(\mathcal{A}) \cup \mathcal{E}(\mathcal{A}_2')$.

As in the cases above, we have to check if there exists a token $\vartheta\downarrow_1 \in \Theta^{\downarrow}$ with $W(\vartheta\downarrow_1) = \langle \mathcal{B}, \mathcal{T}, t\rangle$, such that $(\mathcal{A}_1, s_1) \equiv_k^{MSO} (\mathcal{B}, t)$. If such a $\vartheta\downarrow_1$ exists, we take it. Otherwise we invent a new token $\vartheta\downarrow_1$, add it to $\Theta^{\downarrow}$, and set $W(\vartheta\downarrow_1) := \langle \mathcal{A}_1, \mathcal{S}_1, s_1\rangle$ In any case, we add the following rule to the program $\mathcal{P}$:

$$\begin{aligned}\vartheta\downarrow_1(v_1) \leftarrow\ & bag(v_1, x_0, x_1, \ldots, x_w), child_1(v_1, v), child_2(v_2, v), \vartheta\downarrow(v), \vartheta\uparrow_2(v_2),\\ & bag(v, x_0, x_1, \ldots, x_w), bag(v_2, x_0, x_1, \ldots, x_w).\end{aligned}$$

Now suppose that $\mathcal{S}_1$ is constructed from $\mathcal{S}$ and $\mathcal{S}_2$ by attaching the new node s_1 as second child of s and s_2 as the first child. In this case, the structure $\mathcal{A}_1$ remains exactly the same as in the case above, since the order of the child nodes of a node in the tree decomposition is irrelevant. Thus, whenever the above rule is added to the program $\mathcal{P}$, then also the following rule is added:

$$\vartheta{\downarrow}_1(v_2) \leftarrow bag(v_2, x_0, x_1, \ldots, x_w), child_1(v_1, v), child_2(v_2, v), \vartheta{\downarrow}(v), \vartheta{\uparrow}_2(v_1), bag(v, x_0, x_1, \ldots, x_w), bag(v_1, x_0, x_1, \ldots, x_w).$$

As a result of this top-down construction, the set $\Theta^{\downarrow}$ is a set of tokens $\vartheta{\downarrow}$ representing all possible types ϑ of structures $(\mathcal{B}, t)$, where t is a leaf node of some tree decomposition $\mathcal{T}$ of $\mathcal{B}$. We shall come back to this point at the end of the construction of the desired datalog program.

(3) *Element selection.* We consider all pairs of types $\vartheta{\uparrow}_1 \in \Theta^{\uparrow}$ and $\vartheta{\downarrow}_2 \in \Theta^{\downarrow}$. Let $W(\vartheta{\uparrow}_1) = \langle \mathcal{A}_1, \mathcal{S}_1, s_1 \rangle$ and $W(\vartheta{\downarrow}_2) = \langle \mathcal{A}_2, \mathcal{S}_2, s_2 \rangle$. Moreover, let $A_{s_1} = (a_0, \ldots, a_w)$ and $A_{s_2} = (b_0, \ldots, b_w)$, respectively. We can assume that $dom(\mathcal{A}_1) \cap dom(\mathcal{A}_2) = \emptyset$ because we can replace every witness by a variable disjoint (isomorphic) copy of it.

Let δ be a renaming function with $\delta = \{a_0 \leftarrow b_0, \ldots, a_w \leftarrow b_w\}$. By applying δ to $\langle \mathcal{A}_2, \mathcal{S}_2, s_2 \rangle$, we obtain a new triple $\langle \mathcal{A}'_2, \mathcal{S}'_2, s_2 \rangle$ with $\mathcal{A}'_2 = \mathcal{A}_2\delta$ and $\mathcal{S}'_2 = \mathcal{S}_2\delta$. In particular, we thus have $A_{s_2}\delta = (a_0, \ldots, a_w)$. Clearly, $(\mathcal{A}_2, s_2) \equiv_k^{MSO} (\mathcal{A}'_2, s_2)$ holds.

For every such pair $\langle \mathcal{A}_1, \mathcal{S}_1, s_1 \rangle$ and $\langle \mathcal{A}'_2, \mathcal{S}'_2, s_2 \rangle$, we check if the EDBs are inconsistent, that is, $\mathcal{E}(\mathcal{A}_1) \cap \mathcal{R}(\bar{a}) \neq \mathcal{E}(\mathcal{A}'_2) \cap \mathcal{R}(\bar{a})$. If this is the case, then we ignore this pair. Otherwise, we construct a new tree decomposition $\mathcal{S}$ by identifying s_1 (= the root of $\mathcal{S}_1$) with s_2 (= a leaf of $\mathcal{S}_2$). By construction, $\mathcal{S}$ is a normalized tree decomposition of the structure $\mathcal{A}$ with $dom(A) = dom(\mathcal{A}_1) \cup dom(\mathcal{A}'_2)$ and $\mathcal{E}(\mathcal{A}) = \mathcal{E}(\mathcal{A}_1) \cup \mathcal{E}(\mathcal{A}'_2)$.

Now check for each a_i in $A_{s_1} = A_{s_2}\delta$, if $\mathcal{A} \models \varphi(a_i)$. If this is the case, then we add the following rule to $\mathcal{P}$:

$$\varphi(x_i) \leftarrow \vartheta{\uparrow}_1(v), \vartheta{\downarrow}_2(v), bag(v, x_0, \ldots, x_w).$$

We claim that the program $\mathcal{P}$ with distinguished monadic predicate φ is the desired monadic datalog program, that is, let $\mathcal{A}$ be an arbitrary input τ-structure with tree decomposition $\mathcal{S}$ and let $\mathcal{A}_{td}$ denote the corresponding τ_{td}-structure. Moreover, let $a \in dom(\mathcal{A})$. Then the following equivalence holds: $\mathcal{A} \models \varphi(a)$ if and only if $\varphi(a)$ is true in the minimal model of $\mathcal{P} \cup \mathcal{A}_{td}$.

Note that the intensional predicates in $\Theta^{\uparrow}$, $\Theta^{\downarrow}$, and $\{\varphi\}$ are layered in that we can first evaluate the predicates in $\Theta^{\uparrow}$ over the structure $\mathcal{A}_{td}$, then $\Theta^{\downarrow}$, and finally φ.

The bottom-up construction of $\Theta^{\uparrow}$ guarantees that we indeed construct all possible types of structures $(\mathcal{B}, t)$ with tree decomposition $\mathcal{T}$ and root t. This can be easily shown by Lemma 3.5 and an induction on the size of the tree decomposition $\mathcal{T}$. On the other hand, for every subtree $\mathcal{S}_s$ of $\mathcal{S}$, the type of the induced substructure $\mathcal{I}(\mathcal{A}, \mathcal{S}_s, s)$ is ϑ for some $\vartheta{\uparrow} \in \Theta^{\uparrow}$ if and only if the atom $\vartheta{\uparrow}(s)$ is true in the minimal model of $\mathcal{P} \cup \mathcal{A}_{td}$. Again this can be shown by an easy induction argument using Lemma 3.5.

Analogously, we may conclude via Lemma 3.6 that $\Theta^{\downarrow}$ contains all possible types of structures $(\mathcal{B}, t)$ with tree decomposition $\mathcal{T}$ and some leaf node t. Moreover, for every subtree $\bar{\mathcal{S}}_s$ of $\mathcal{S}$, the type of the induced substructure $\mathcal{I}(\mathcal{A}, \bar{\mathcal{S}}_s, s)$ is ϑ for some $\vartheta{\downarrow} \in \Theta^{\downarrow}$ if and only if the atom $\vartheta{\downarrow}(s)$ is true in the minimal model of $\mathcal{P} \cup \mathcal{A}_{td}$. The definition of the predicate φ in part 3 is a direct realization of Lemma 3.7. It thus follows that $\mathcal{A} \models \varphi(a)$ if and only if $\varphi(a)$ is true in the minimal model of $\mathcal{P} \cup \mathcal{A}_{td}$.

Finally, an inspection of all datalog rules added to $\mathcal{P}$ by this construction shows that these rules are indeed quasiguarded, that is, they all contain an atom B with an extensional predicate, such that all other variables in this rule are functionally dependent on the variables in B. For instance, in the rule added to $\Theta^{\uparrow}$ in the case of a branch node, the atom $bag(v, x_0, \ldots, x_w)$ is the quasiguard. Indeed, the remaining variables v_1 and v_2 in this rule are functionally dependent on v via the atoms $child_1(v_1, v)$and $child_2(v_2, v)$. □

Above all, Theorem 4.6 is an expressivity result. However, it can of course be used to derive also a complexity result. Indeed, we can state a slightly extended version of Courcelle's Theorem as a corollary (which is in turn a special case of Theorem 4.12 in Flum et al. [2002]).

COROLLARY **4.7** *The evaluation problem of unary MSO-queries $\varphi(x)$ over τ-structures $\mathcal{A}$ of treewidth w can be solved in time $\mathcal{O}(f(|\varphi(x)|, w) * |\mathcal{A}|)$ for some function f.*

PROOF. Suppose that we are given an MSO query $\varphi(x)$ and some treewidth w. By Theorem 4.6, we can construct an equivalent, quasiguarded datalog program $\mathcal{P}$. The whole construction is independent of the data. Hence, the time for this construction and the size of $\mathcal{P}$ are both bounded by some term $f(|\varphi(x)|, w)$. By Bodlaender [1996], a tree decomposition $\mathcal{T}$ of $\mathcal{A}$ and, therefore, also the extended structure $\mathcal{A}_{td}$ can be computed in time $\mathcal{O}(|\mathcal{A}|)$. Finally, by Theorem 4.5, the quasiguarded program $\mathcal{P}$ can be evaluated over $\mathcal{A}_{td}$ in time $\mathcal{O}(|\mathcal{P}| * |\mathcal{A}_{td}|)$, from which the desired overall time bound follows. □

As with Courcelle's Theorem recalled in Section 2.3, this linear upper bound on the complexity does not immediately give a feasible algorithm due to huge multiplicative constants. In fact, an algorithm using directly the construction from the proof of Theorem 4.6 would end up with a constant of nonelementary size (with respect to the quantifier depth of the formula φ). We shall see in Section 5 that it is nonetheless possible to come up with feasible algorithms (with singly exponential constants for some NP-complete problems) by appropriately adapting the monadic datalog approach from this section.

Note that Theorem 4.6 is, of course, not only applicable to MSO-definable *unary queries* but also to 0-*ary queries*, that is, MSO queries defining a decision problem. An inspection of the proof of Theorem 4.6 reveals that several simplifications are possible in this case. Above all, the whole "top-down" construction of $\Theta^{\downarrow}$ can be omitted. Moreover, the rules with head predicate φ are now much simpler: let φ be a 0-ary MSO formula and let $\Theta^{\uparrow}$ denote the set of types obtained by the "bottom-up" construction in the above proof. Then we

define $\Theta_0^{\uparrow} = \{\vartheta\uparrow \mid W(\vartheta\uparrow) = \langle \mathcal{A}, \mathcal{S}, s\rangle$ and $\mathcal{A} \models \varphi\}$. Finally, we add the following set of rules with head predicate φ to our datalog program:

$$\varphi \leftarrow root(v), \vartheta\uparrow_0(v).$$

for every $\vartheta\uparrow_0 \in \Theta_0^{\uparrow}$. We shall make use of these simplifications in Section 5.1 and 5.2 when we present new algorithms for two decision problems. In contrast, these simplifications are no longer possible when we consider an enumeration problem in Section 5.3. In particular, the "top-down" construction will indeed be required then.

5. METAPROGRAMS BASED ON MONADIC DATALOG

We now put our monadic datalog approach to work by constructing several new algorithms. We start off with a simple example, namely, the 3-Colorability problem, which will help to illustrate the basic ideas; see Section 5.1. Our ultimate goal is to tackle two more involved problems; namely the PRIMALITY decision problem and the PRIMALITY enumeration problem; see Sections 5.2 and 5.3. All these problems are well-known to be intractable. However, since they are expressible in MSO over appropriate structures, they are fixed-parameter tractable with respect to the treewidth. In this section, we show that these problems admit succinct and efficient solutions via datalog.

Recall from Section 2.2 that we now consider tree decompositions in the normal form according to Definition 2.6. Hence, bags are considered as sets (rather than tuples) and, apart from the leaf nodes, we distinguish three kinds of internal nodes, namely, *element removal nodes* (which remove one domain element from the bag of the child node), *element introduction nodes* (which introduce one new element), and *branch nodes* (which have two child nodes—each with identical bag as the parent).

5.1 The 3-Colorability Problem

Suppose that a graph (V, E) with vertices V and edges E is given as a τ-structure with $\tau = \{e\}$, that is, e is the binary edge relation. This graph is 3-colorable if and only if there exists a partition of V into three sets $\mathcal{R}$, $\mathcal{G}$, $\mathcal{B}$, such that no two adjacent vertices $v_1, v_2 \in V$ are in the same set $\mathcal{R}$, $\mathcal{G}$, or $\mathcal{B}$. This criterion can be easily expressed by an MSO-sentence, namely,

$$\begin{aligned}\varphi \equiv\ & \exists R \exists G \exists B[Partition(R, G, B) \wedge \forall v_1 \forall v_2[e(v_1, v_2) \rightarrow \\ & (\neg R(v_1) \vee \neg R(v_2)) \wedge (\neg G(v_1) \vee \neg G(v_2)) \wedge (\neg B(v_1) \vee \neg B(v_2))]]\end{aligned}$$

with

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

Suppose that a graph (V, E) together with a tree decomposition $\mathcal{T}$ of width w is given as a τ_{td}-structure with $\tau_{td} = \{e, root, leaf, child_1, child_2, bag\}$. In Figure 5, we describe a datalog program which takes such a τ_{td}-structure as input and decides if the graph thus represented is 3-colorable.

Program 3-Colorability

/* leaf node. */
$solve(s, R, G, B) \leftarrow leaf(s)$, $bag(s, X)$, $partition(s, R, G, B)$, $allowed(s, R)$, $allowed(s, G)$, $allowed(s, B)$.

/* element introduction node. */
$solve(s, R \uplus \{v\}, G, B) \leftarrow bag(s, X \uplus \{v\})$, $child_1(s_1, s)$, $bag(s_1, X)$, $solve(s_1, R, G, B)$, $allowed(s, R \uplus \{v\})$.
$solve(s, R, G \uplus \{v\}, B) \leftarrow bag(s, X \uplus \{v\})$, $child_1(s_1, s)$, $bag(s_1, X)$, $solve(s_1, R, G, B)$, $allowed(s, G \uplus \{v\})$.
$solve(s, R, G, B \uplus \{v\}) \leftarrow bag(s, X \uplus \{v\})$, $child_1(s_1, s)$, $bag(s_1, X)$, $solve(s_1, R, G, B)$, $allowed(s, B \uplus \{v\})$.

/* element removal node. */
$solve(s, R, G, B) \leftarrow bag(s, X)$, $child_1(s_1, s)$, $bag(s_1, X \uplus \{v\})$, $solve(s_1, R \uplus \{v\}, G, B)$.
$solve(s, R, G, B) \leftarrow bag(s, X)$, $child_1(s_1, s)$, $bag(s_1, X \uplus \{v\})$, $solve(s_1, R, G \uplus \{v\}, B)$.
$solve(s, R, G, B) \leftarrow bag(s, X)$, $child_1(s_1, s)$, $bag(s_1, X \uplus \{v\})$, $solve(s_1, R, G, B \uplus \{v\})$.

/* branch node. */
$solve(s, R, G, B) \leftarrow bag(s, X)$, $child_1(s_1, s)$, $child_2(s_2, s)$, $bag(s_1, X)$, $bag(s_2, X)$, $solve(s_1, R, G, B)$, $solve(s_2, R, G, B)$.

/* result (at the root node). */
$success \leftarrow root(s)$, $solve(s, R, G, B)$.

Fig. 5. 3-Colorability test.

Some words on the notation used in this program are in order: we are using lowercase letters s and v (possibly with subscripts) as datalog variables for a single node in $\mathcal{T}$ and for a single vertex in V, respectively. In contrast, uppercase letters X, R, G, and B are used as datalog variables denoting sets of vertices. Note that these sets are not sets in the general sense, since their cardinality is restricted by the size $w + 1$ of the bags, where w is a fixed constant. Hence, these "fixed-size" sets can be simply implemented by means of k-tuples with $k \leq (w + 1)$ over $\{0, 1\}$. For the sake of readability, we are using nondatalog expressions with the set operator $\uplus$ (disjoint union). For the fixed-size sets under consideration here, one could, of course, replace this operator by pure datalog expressions as we shall explain later in this section.

It is convenient to introduce the following notation. Let $G = (V, E)$ be the input graph with tree decomposition $\mathcal{T}$. For any node s in $\mathcal{T}$, we write as usual $\mathcal{T}_s$ to denote, the subtree. of $\mathcal{T}$ rooted at s. Moreover, we write $V(s)$ and $V(\mathcal{T}_s)$ to denote, respectively, the set of vertices of the bag of s and the union of those sets associated with the nodes in $\mathcal{T}_s$.

Our 3-Colorability-program checks if G is 3-colorable via the criterion mentioned above, that is, there exists a partition of V into three sets $\mathcal{R}$, $\mathcal{G}$, $\mathcal{B}$, such that no two adjacent vertices $v_1, v_2 \in V$ are in the same set $\mathcal{R}$, $\mathcal{G}$, or $\mathcal{B}$.

At the heart of this program is the intensional predicate *solve*(s, R, G, B) with the following intended meaning: s denotes a node in $\mathcal{T}$ and R, G, B are the intersections of $\mathcal{R}$, $\mathcal{G}$, $\mathcal{B}$ with $V(s)$. For all values s, R, G, B, the ground fact *solve*$(s, R, G, B,)$ shall be true in the minimal model of the program and the input structure if and only if the following condition holds:

Auxiliary Predicates

$partition(s, R, G, B) \leftarrow bag(s, X), R \cup G \cup B = X, R \cap G = \emptyset, R \cap B = \emptyset, G \cap B = \emptyset.$
$allowed(s, Y) \leftarrow bag(s, X), Y \subseteq X, \neg forbidden(s, Y).$
$forbidden(s, Y) \leftarrow bag(s, X), Y \subseteq X, v_1 \in Y, v_2 \in Y, e(v_1, v_2).$

Fig. 6. Auxiliary predicates *partition* and *allowed*.

PROPERTY A. There exist extensions $\hat{R}$ of R, $\hat{G}$ of G, and $\hat{B}$ of B to $V(\mathcal{T}_s)$, such that

(1) $\hat{R}$, $\hat{G}$, and $\hat{B}$ form a partition of $V(\mathcal{T}_s)$ and
(2) no two adjacent vertices $v_1, v_2 \in V(\mathcal{T}_s)$ are in the same set $\hat{R}$, $\hat{G}$, or $\hat{B}$.

In other words, $\hat{R}$, $\hat{G}$, and $\hat{B}$ is a valid 3-coloring of the vertices in $V(\mathcal{T}_s)$ and R, G, and B are the intersections of $\hat{R}$, $\hat{G}$, and $\hat{B}$ with $V(s)$.

The main task of the program is the computation of all facts *solve*(s, R, G, B) via a bottom-up traversal of the tree decomposition. The other predicates have the following meaning (see Figure 6):

—*partition*(s, R, G, B) is true in the minimal model if and only if R, G, B is a partition of the bag X at node s in the tree decomposition.
—*allowed*(s, X) is true in the minimal model if and only if X contains no adjacent vertices v_1, v_2.

Recall that the cardinality of the sets X, R, G, B occurring as arguments of *partition* and *allowed* is bounded by the fixed constant $w + 1$. In fact, both the *partition* predicate and the *allowed* predicate could also be treated as extensional predicates by computing all facts *partition*(s, R, G, B) and *allowed*(s, X) for each node s in $\mathcal{T}$ as part of the computation of the tree decomposition.

The intuition of the rules with the *solve* predicate in the head is now clear: at the *leaf nodes*, the program generates ground facts *solve*(s, R, G, B) for all possible partitions of the bag X at s, such that none of the sets R, G, B contains two adjacent vertices. The three rules for *element introduction nodes* distinguish the three cases if the new vertex v is added to R, G, or B, respectively. Of course, by the *allowed* atom in the body of these three rules, the attempt to add v to any of the sets R, G, or B may fail. The three rules for *element removal nodes* distinguish the three cases if the removed vertex was in R, G, or B, respectively. The rule for *branch nodes* combines *solve* facts with identical values of (R, G, B) at the child nodes s_1 and s_2 to the corresponding *solve* fact at s.

In summary, the 3-colorability-program has the following properties.

THEOREM 5.1. *The datalog program in Figure* 5 *decides the* 3*-Colorability problem, that is, the fact "success" is true in the minimal model of this program and the input* τ_{td}*-structure* $\mathcal{A}_{td}$ *if and only if* $\mathcal{A}_{td}$ *encodes a 3-colorable graph* (V, E) *together with a tree decomposition* $\mathcal{T}$ *of* (V, E)*. Moreover, for any graph* (V, E) *and tree decomposition* $\mathcal{T}$ *of width at most* w*, the program can be evaluated in time* $\mathcal{O}(3^w * |(V, E)|)$*.*

PROOF. By the above considerations, it is clear that the predicate *solve* indeed has the meaning described by Property A. A formal proof of this fact by structural induction on $\mathcal{T}$ is immediate and therefore omitted here. Then the rule with head *success* reads as follows: *success* is true in the minimal model if and only if s denotes the root of $\mathcal{T}$ and there exist extensions $\hat{R}$, $\hat{G}$, and $\hat{B}$ of R, G, B to $V(\mathcal{T}_s)$ (which is identical to V in case of the root node s), such that $\hat{R}$, $\hat{G}$, and $\hat{B}$ is a valid 3-coloring of the vertices in $V(\mathcal{T}_s) = V$.

For the upper bound on the time complexity, we observe that in all facts $solve(s, R, G, B)$ derived by the program, the sets R, G, B form a partition of the bag at s (which contains at most $w + 1$ vertices). Again, this property can be easily proved by structural induction on $\mathcal{T}$. Hence, the datalog program $\mathcal{P}$ in Figure 5 is equivalent to a ground program $\mathcal{P}'$ where each rule of $\mathcal{P}$ is replaced by $\mathcal{O}(3^w * |(V, E)|)$ ground rules. These ground rules can be computed as follows: at a leaf node s, we have 3^{w+1} possible values of (R, G, B), such that $partition(s, R, G, B)$ is fulfilled. Moreover, for the bag at s, there are in total at most 2^{w+1} possible instances of the $allowed(s, X)$ predicate, which can be computed in $\mathcal{O}(2^w * w^2)$, that is, we have to check for every pair of vertices (x, y) in a set of at most $w + 1$ vertices, that x and y are not adjacent.

Likewise, for all other kinds of nodes, there are at most 3^{w+1} possible values of (R, G, B), such that $solve(s, R, G, B)$ is the head of a rule in the ground program $\mathcal{P}'$. Note that, for all rules in Figure 5 at element introduction and branch nodes, the instantiation of the variables (s, R, G, B) in the head admits at most one instantiation of the variables in the body. If s is an element removal node, then the bag at s has at most w elements and there are only 3^w possible values of (R, G, B), such that $solve(s, R, G, B)$ is the head of a ground rule in $\mathcal{P}'$. On the other hand, each such value (s, R, G, B) admits three possible instantiations of the variables in the body. Hence, we again end up with 3^{w+1} possible ground rules in $\mathcal{P}'$. Finally, also the rule with the *success*-predicate in the head admits at most 3^{w+1} possible ground instantiations (of the *solve* predicate in the body and, hence, of the entire rule).

In total, we can evaluate the program $\mathcal{P}$ in Figure 5 over an input graph (V, E) by first computing the equivalent ground program $\mathcal{P}'$ with $\mathcal{O}(3^w * |(V, E)|)$ rules and then evaluating $\mathcal{P}'$ in linear time. □

Actually, the linear time data complexity in Theorem 5.1 could also be seen as follows (without getting the concrete value 3^w for the multiplicative constant though): our program in Figure 5 is essentially a succinct representation of a quasiguarded monadic datalog program. For instance, in the atom $solve(s, R, G, B)$, the sets R, G, B are subsets of the bag of s. Hence, each combination R, G, B could be represented by three subsets r_1, r_2, r_3 over $\{0, \ldots, w\}$ referring to indices of elements in the bag of s. Recall that w is a fixed constant. Hence, $solve(s, R, G, B)$ is simply a succinct representation of constantly many monadic predicates of the form $solve_{\langle r_1, r_2, r_3 \rangle}(s)$. The quasiguard in each rule can thus be any atom with argument s, for example, $bag(s, X)$ or $bag(s, X) \uplus \{v\}$. Thus, an upper bound of the form $\mathcal{O}(f(w) * |(V, E)|)$ for some function f depending on the treewidth w of the graph (but not on the size the graph) follows immediately from Theorem 4.5.

Discussion. Let us briefly compare the monadic program constructed in the proof of Theorem 4.6 with the 3-Colorability program in Figure 5. Actually, since we are dealing with a decision problem here, we only look at the bottom-up construction in the proof of Theorem 4.6; the top-down construction is not needed for a 0-ary target formula $\varphi()$. As was already mentioned above, the atoms $solve(s, R, G, B)$ can be thought of as a succinct representation for atoms of the form $solve_{\langle r_1,r_2,r_3\rangle}(s)$. Now the question naturally arises where the type ϑ of some node s from the proof of Theorem 4.6 is present in the 3-Colorability program. A first tentative answer is that this type essentially corresponds to the set $R(s) = \{\langle r_1, r_2, r_3\rangle \mid solve_{\langle r_1,r_2,r_3\rangle}(s)$ is true in the minimal model$\}$. However, there are two significant aspects which distinguish our 3-Colorability program from merely a succinct representation of the type transitions encoded in the monadic datalog program of Theorem 4.6:

(1) By Property A, we are only interested in the types of those structures which—in principle—could be extended in bottom-up direction to a 3-colorable graph. Hence, in contrast to the construction in the proof of Theorem 4.6, our 3-Colorability program does clearly not keep track of all possible types that the substructure induced by some tree decomposition $\mathcal{T}_s$ may possibly have.

(2) $R(s) = \{\langle r_1, r_2, r_3\rangle \mid solve_{\langle r_1,r_2,r_3\rangle}(s)$ is true in the minimal model$\}$ does not exactly correspond to the type of s. Instead, it only describes the crucial properties of the type. Thus, the 3-Colorability program somehow "aggregates" several types from the proof of Theorem 4.6.

These two properties ensure that the 3-Colorability program is much shorter than the program in the proof of Theorem 4.6 and that the difference between these two programs is not just due to the succinct representation of a monadic program by a nonmonadic one. The rationale behind this improvement is that we take the target MSO formula φ (namely, the characterization of 3-Colorability) into account for the entire construction of the datalog program in Figure 5. In contrast, the datalog program constructed in the proof of Theorem 4.6 is fully generic, that is, the rules describing all possible type-transitions in the proof of Theorem 4.6 only depend on the treewidth w and the quantifier depth k but not on the concrete target MSO formula φ that we ultimately want to evaluate.

5.1.2 *Set Arithmetic Versus Pure Datalog.* The program shown in Figure 5 contains set variables and set operations which are not part of the datalog language. For given treewidth w, it is fairly straightforward to transform the set arithmetic into pure datalog constructs. Below, we instantiate the 3-Colorability program as a pure datalog program for treewidth 3; see Figure 7. Note, however, that there are state-of-the-art datalog engines which actually do support sets. For instance, a recent extension of the DLV-system [Leone et al. 2006], which is called *DLV-Complex*,[1] provides special built-in predicates for set arithmetic.

[1] `http://www.mat.unical.it/dlv-complex`.

Program 3-Colorability, Datalog Instantiation for Treewidth = 3.

/* leaf node. */

$$solve(s, r_1, r_2, r_3, r_4, g_1, g_2, g_3, g_4, b_1, b_2, b_3, b_4) \leftarrow$$
$$\quad leaf(s),\ partition(s, r_1, r_2, r_3, r_4, g_1, g_2, g_3, g_4, b_1, b_2, b_3, b_4),$$
$$\quad allowed(s, r_1, r_2, r_3, r_4),\ allowed(s, g_1, g_2, g_3, g_4),\ allowed(s, b_1, b_2, b_3, b_4).$$

/* element introduction node. */

$$solve(s, r_1, r_2, r_3, v, g_1, g_2, g_3, g_4, b_1, b_2, b_3, b_4) \leftarrow$$
$$\quad introduced(s, v),\ child_1(s_1, s),\ allowed(s, r_1, r_2, r_3, v),$$
$$\quad solve(s_1, r_1, r_2, r_3, \bot, g_1, g_2, g_3, g_4, b_1, b_2, b_3, b_4),\ r_1 \neq \bot,\ r_2 \neq \bot,\ r_3 \neq \bot.$$
$$solve(s, r_1, r_2, v, \bot, g_1, g_2, g_3, g_4, b_1, b_2, b_3, b_4) \leftarrow$$
$$\quad introduced(s, v),\ child_1(s_1, s),\ allowed(s, r_1, r_2, v, \bot),$$
$$\quad solve(s_1, r_1, r_2, \bot, \bot, g_1, g_2, g_3, g_4, b_1, b_2, b_3, b_4),\ r_1 \neq \bot,\ r_2 \neq \bot.$$
$$solve(s, r_1, v, \bot, \bot, g_1, g_2, g_3, g_4, b_1, b_2, b_3, b_4) \leftarrow$$
$$\quad introduced(s, v),\ child_1(s_1, s),\ allowed(s, r_1, v, \bot, \bot),$$
$$\quad solve(s_1, r_1, \bot, \bot, \bot, g_1, g_2, g_3, g_4, b_1, b_2, b_3, b_4),\ r_1 \neq \bot.$$
$$solve(s, v, \bot, \bot, \bot, g_1, g_2, g_3, g_4, b_1, b_2, b_3, b_4) \leftarrow$$
$$\quad introduced(s, v),\ child_1(s_1, s),\ solve(s_1, \bot, \bot, \bot, \bot, g_1, g_2, g_3, g_4, b_1, b_2, b_3, b_4).$$
$$solve(s, r_1, r_2, r_3, r_4, g_1, g_2, g_3, v, b_1, b_2, b_3, b_4) \leftarrow$$
$$\quad introduced(s, v),\ child_1(s_1, s),\ allowed(s, g_1, g_2, g_3, v),$$
$$\quad solve(s_1, r_1, r_2, r_3, r_4, g_1, g_2, g_3, \bot, b_1, b_2, b_3, b_4),\ g_1 \neq \bot,\ g_2 \neq \bot,\ g_3 \neq \bot.$$

$$\vdots$$

$$solve(s, r_1, r_2, r_3, r_4, g_1, g_2, g_3, g_4, v, \bot, \bot, \bot) \leftarrow introduced(s, v),\ child_1(s_1, s),$$
$$\quad solve(s_1, r_1, r_2, r_3, r_4, g_1, g_2, g_3, g_4, \bot, \bot, \bot, \bot).$$

/* element removal node. */

$$solve(s, r_1, r_2, r_3, \bot, g_1, g_2, g_3, g_4, b_1, b_2, b_3, b_4) \leftarrow removed(s, v),\ child_1(s_1, s),$$
$$\quad solve\_permutation(s_1, r_1, r_2, r_3, v, g_1, g_2, g_3, g_4, b_1, b_2, b_3, b_4).$$
$$solve(s, r_1, r_2, \bot, \bot, g_1, g_2, g_3, g_4, b_1, b_2, b_3, b_4) \leftarrow removed(s, v),\ child_1(s_1, s),$$
$$\quad solve\_permutation(s_1, r_1, r_2, v, \bot, g_1, g_2, g_3, g_4, b_1, b_2, b_3, b_4).$$
$$solve(s, r_1, \bot, \bot, \bot, g_1, g_2, g_3, g_4, b_1, b_2, b_3, b_4) \leftarrow removed(s, v),\ child_1(s_1, s),$$
$$\quad solve\_permutation(s_1, r_1, v, \bot, \bot, g_1, g_2, g_3, g_4, b_1, b_2, b_3, b_4).$$
$$solve(s, \bot, \bot, \bot, \bot, g_1, g_2, g_3, g_4, b_1, b_2, b_3, b_4) \leftarrow removed(s, v),\ child_1(s_1, s),$$
$$\quad solve\_permutation(s_1, v, \bot, \bot, \bot, g_1, g_2, g_3, g_4, b_1, b_2, b_3, b_4).$$
$$solve(s, r_1, r_2, r_3, r_4, g_1, g_2, g_3, \bot, b_1, b_2, b_3, b_4) \leftarrow removed(s, v),\ child_1(s_1, s),$$
$$\quad solve\_permutation(s_1, r_1, r_2, r_3, r_4, g_1, g_2, g_3, v, b_1, b_2, b_3, b_4).$$

$$\vdots$$

$$solve(s, r_1, r_2, r_3, r_4, g_1, g_2, g_3, g_4, \bot, \bot, \bot, \bot) \leftarrow removed(s, v),\ child_1(s_1, s),$$
$$\quad solve\_permutation(s_1, r_1, r_2, r_3, r_4, g_1, g_2, g_3, g_4, v, \bot, \bot, \bot).$$

/* branch node. */

$$solve(s, r_1, r_2, r_3, r_4, g_1, g_2, g_3, g_4, b_1, b_2, b_3, b_4) \leftarrow child_1(s_1, s),\ child_2(s_2, s),$$
$$\quad solve(s_1, r_1, r_2, r_3, r_4, g_1, g_2, g_3, g_4, b_1, b_2, b_3, b_4),$$
$$\quad solve\_permutation(s_2, r_1, r_2, r_3, r_4, g_1, g_2, g_3, g_4, b_1, b_2, b_3, b_4).$$

/* result (at the root node). */

$$success \leftarrow root(s),\ solve(s, r_1, r_2, r_3, r_4, g_1, g_2, g_3, g_4, b_1, b_2, b_3, b_4).$$

Fig. 7. Datalog instantiation of the 3-Colorability program.

Let w denote some constant which is an upper bound on the treewidth of the intended inputs. The basic idea of the transformation of the program in Figure 5 into a pure datalog program is to replace each set variable X with $w + 1$ individual variables. In Figure 7, we show the resulting pure datalog program for treewidth $w = 3$. Of course, a set may contain fewer than four elements. In this case, we pad the set with appropriately many copies of the auxiliary symbol $\perp$. For instance, a set $X = \{v_1, v_2\}$ can be represented by the quadruple $v_1, v_2, \perp, \perp$. Note that in Figure 7 (and also for the definition of auxiliary predicates in Figures 8 and 9) we adopt the convention that the symbol $\perp$ is filled in from the right, for example, $v_1, \perp, v_2\perp$ would not be a legal representation of $X = \{v_1, v_2\}$.

The predicates *solve* and *partition*, which contain three sets, now have 12 positions (in addition to the argument s) to represent these three sets, for example, the intended meaning of $solve(s, r_1, r_2, r_3, r_4, g_1, g_2, g_3, g_4, b_1, b_2, b_3, b_4)$ is that the sets R, G, B have the values $R = \{r_1, r_2, r_3, r_4\} \setminus \{\perp\}$, $G = \{g_1, g_2, g_3, g_4\} \setminus \{\perp\}$, and $B = \{b_1, b_2, b_3, b_4\} \setminus \{\perp\}$.

Of course, things are now slightly complicated by the fact that we can choose different orderings to arrange the elements of a set as a list of individual variables. Hence, we define additional auxiliary predicates *bag_permutation* and *solve_permutation* which generate all those facts that can be obtained from the *bag*—respectively, *solve*—predicate by permuting the elements (see Figures 8 and 9).

The datalog program in Figure 7 uses the auxiliary predicates $partition(s, r_1, r_2, r_3, r_4, g_1, g_2, g_3, g_4, b_1, b_2, b_3, b_4)$, $allowed(s, v_1, v_2, v_3, v_4)$, $introduced(s, v)$, $removed(s, v)$, and $solve\_permutation(s, r_1, r_2, r_3, r_4, g_1, g_2, g_3, g_4, b_1, b_2, b_3, b_4)$ with the obvious meaning; for example: $introduced(s, v)$ means that s is an element introduction node, such that v is the newly introduced vertex (i.e., the vertex which was not present in the bag at the child node of s). Likewise, $removed(s, v)$ means that s is an element removal node, such that v is the removed vertex (i.e., the vertex which was present in the bag at the child node of s but not in the bag of s). The definition of these auxiliary predicates is shown in Figure 8. There, two additional auxiliary predicates *bag_permutation* and *bag_subset* are needed, whose meaning is also obvious, namely: $bag\_permutation(s, v_1, v_2, v_3, v_4)$ means that v_1, v_2, v_3, v_4 is a permutation of the vertices in the bag at node s; $bag\_subset(s, v_1, v_2, v_3, v_4)$ means that v_1, v_2, v_3, v_4 represents a subset of the vertices in the bag at node s. As mentioned above, a subset of cardinality smaller than 4 is represented by filling in the symbol $\perp$ appropriately often (in contiguous places starting from the right). The details are worked out in Figure 9. The number of resulting rules in Figures 7, 8, and 9 is (singly) exponential with respect to the treewidth w.

5.1.3 *Computing a Solution.* We conclude this section by sketching an algorithm that computes one possible solution to the 3-Coloring problem in the case that the *success*-fact is true in the minimal model of the program in Figure 5. To this end, we construct a recursive procedure which takes as input a tuple (s, R, G, B) and accumulates a possible 3-coloring in global variables $\hat{R}$,

Auxiliary Predicates: Datalog Instantiation for Treewidth = 3.

/* *allowed* */
$allowed(s, v_1, v_2, v_3, v_4) \leftarrow bag\_subset(s, v_1, v_2, v_3, v_4),$
$\neg e(v_1, v_2), \neg e(v_1, v_3), \neg e(v_1, v_4), \neg e(v_2, v_3), \neg e(v_2, v_4), \neg e(v_3, v_4).$

/* *partition* */
$partition(s, v_1, v_2, v_3, v_4, \bot, \bot, \bot, \bot, \bot, \bot, \bot, \bot) \leftarrow bag\_permutation(s, v_1, v_2, v_3, v_4).$
$partition(s, v_1, v_2, v_3, \bot, v_4, \bot, \bot, \bot, \bot, \bot, \bot, \bot) \leftarrow bag\_permutation(s, v_1, v_2, v_3, v_4).$
$partition(s, v_1, v_2, v_3, \bot, \bot, \bot, \bot, \bot, v_4, \bot, \bot, \bot) \leftarrow bag\_permutation(s, v_1, v_2, v_3, v_4).$
$partition(s, v_1, v_2, \bot, \bot, v_3, v_4, \bot, \bot, \bot, \bot, \bot, \bot) \leftarrow bag\_permutation(s, v_1, v_2, v_3, v_4).$
$partition(s, v_1, v_2, \bot, \bot, v_3, \bot, \bot, \bot, v_4, \bot, \bot, \bot) \leftarrow bag\_permutation(s, v_1, v_2, v_3, v_4).$
$partition(s, v_1, v_2, \bot, \bot, \bot, \bot, \bot, \bot, v_3, v_4, \bot, \bot) \leftarrow bag\_permutation(s, v_1, v_2, v_3, v_4).$
$partition(s, v_1, \bot, \bot, \bot, v_2, v_3, v_4, \bot, \bot, \bot, \bot, \bot) \leftarrow bag\_permutation(s, v_1, v_2, v_3, v_4).$
$\vdots$
$partition(s, v_1, \bot, \bot, \bot, \bot, \bot, \bot, \bot, v_2, v_3, v_4, \bot) \leftarrow bag\_permutation(s, v_1, v_2, v_3, v_4).$
$\vdots$
$partition(s, \bot, \bot, \bot, \bot, \bot, \bot, \bot, \bot, v_1, v_2, v_3, v_4) \leftarrow bag\_permutation(s, v_1, v_2, v_3, v_4).$

/* *introduced* */
$introduced(s, v) \leftarrow bag\_permutation(s, v_1, v_2, v_3, v), child_1(s_1, s),$
$bag\_permutation(s_1, v_1, v_2, v_3, \bot).$

/* *removed* */
$removed(s, v) \leftarrow bag\_permutation(s, v_1, v_2, v_3, \bot), child_1(s_1, s),$
$bag\_permutation(s_1, v_1, v_2, v_3, v).$

/* *solve_permutation* */
$solve\_permutation(s, r_1, r_2, r_3, r_4, \bot, \bot, \bot, \bot, \bot, \bot, \bot, \bot) \leftarrow$
$solve(s, r_1, r_2, r_3, r_4, \bot, \bot, \bot, \bot, \bot, \bot, \bot, \bot).$
$solve\_permutation(s, r_1, r_2, r_4, r_3, \bot, \bot, \bot, \bot, \bot, \bot, \bot, \bot) \leftarrow$
$\vdots$ $solve(s, r_1, r_2, r_3, r_4, \bot, \bot, \bot, \bot, \bot, \bot, \bot, \bot), r_4 \neq \bot.$
$solve\_permutation(s, r_4, r_3, r_2, r_1, \bot, \bot, \bot, \bot, \bot, \bot, \bot, \bot) \leftarrow$
$solve(s, r_1, r_2, r_3, r_4, \bot, \bot, \bot, \bot, \bot, \bot, \bot, \bot), r_4 \neq \bot.$
$solve\_permutation(s, r_1, r_2, r_3, \bot, g_1, \bot, \bot, \bot, \bot, \bot, \bot, \bot) \leftarrow$
$solve(s, r_1, r_2, r_3, \bot, g_1, \bot, \bot, \bot, \bot, \bot, \bot, \bot).$
$solve\_permutation(s, r_1, r_3, r_2, \bot, g_1, \bot, \bot, \bot, \bot, \bot, \bot, \bot) \leftarrow$
$\vdots$ $solve(s, r_1, r_2, r_3, \bot, g_1, \bot, \bot, \bot, \bot, \bot, \bot, \bot), r_3 \neq \bot.$
$solve\_permutation(s, r_1, r_2, \bot, \bot, g_1, g_2, \bot, \bot, \bot, \bot, \bot, \bot) \leftarrow$
$solve(s, r_1, r_2, \bot, \bot, g_1, g_2, \bot, \bot, \bot, \bot, \bot, \bot).$
$solve\_permutation(s, r_1, r_2, \bot, \bot, g_2, g_1, \bot, \bot, \bot, \bot, \bot, \bot) \leftarrow$
$solve(s, r_1, r_2, \bot, \bot, g_1, g_2, \bot, \bot, \bot, \bot, \bot, \bot), g_2 \neq \bot.$
$solve\_permutation(s, r_2, r_1, \bot, \bot, g_1, g_2, \bot, \bot, \bot, \bot, \bot, \bot) \leftarrow$
$solve(s, r_1, r_2, \bot, \bot, g_1, g_2, \bot, \bot, \bot, \bot, \bot, \bot), r_2 \neq \bot.$
$solve\_permutation(s, r_2, r_1, \bot, \bot, g_2, g_1, \bot, \bot, \bot, \bot, \bot, \bot) \leftarrow$
$\vdots$ $solve(s, r_1, r_2, \bot, \bot, g_1, g_2, \bot, \bot, \bot, \bot, \bot, \bot), r_2 \neq \bot, g_2 \neq \bot.$
$solve\_permutation(s, \bot, \bot, \bot, \bot, \bot, \bot, \bot, \bot, b_4, b_3, b_2, b_1) \leftarrow$
$solve(s, \bot, \bot, \bot, \bot, \bot, \bot, \bot, \bot, b_1, b_2, b_3, b_4), b_4 \neq \bot.$

Fig. 8. Datalog instantiation of the auxiliary predicates.

Further Auxiliary Predicates: *bag_permutation* and *bag_subset* for Treewidth = 3.

/* *bag_permutation* */
$bag\_permutation(s, v_1, v_2, v_3, v_4) \leftarrow bag(s, v_1, v_2, v_3, v_4).$
$bag\_permutation(s, v_1, v_2, v_4, v_3) \leftarrow bag(s, v_1, v_2, v_3, v_4), v_4 \neq \bot.$
$bag\_permutation(s, v_1, v_3, v_2, v_4) \leftarrow bag(s, v_1, v_2, v_3, v_4), v_3 \neq \bot.$
$bag\_permutation(s, v_1, v_3, v_4, v_2) \leftarrow bag(s, v_1, v_2, v_3, v_4), v_4 \neq \bot.$
$bag\_permutation(s, v_1, v_4, v_2, v_3) \leftarrow bag(s, v_1, v_2, v_3, v_4), v_4 \neq \bot.$
$bag\_permutation(s, v_1, v_4, v_3, v_2) \leftarrow bag(s, v_1, v_2, v_3, v_4), v_4 \neq \bot.$
$bag\_permutation(s, v_2, v_1, v_3, v_4) \leftarrow bag(s, v_1, v_2, v_3, v_4), v_2 \neq \bot.$
$\vdots$
$bag\_permutation(s, v_4, v_3, v_2, v_1) \leftarrow bag(s, v_1, v_2, v_3, v_4), v_4 \neq \bot.$

/* *bag_subset* */
$bag\_subset(s, v_1, v_2, v_3, v_4) \leftarrow bag\_permutation(s, v_1, v_2, v_3, v_4).$
$bag\_subset(s, v_1, v_2, v_3, \bot) \leftarrow bag\_permutation(s, v_1, v_2, v_3, v_4).$
$bag\_subset(s, v_1, v_2, \bot, \bot) \leftarrow bag\_permutation(s, v_1, v_2, v_3, v_4).$
$bag\_subset(s, v_1, \bot, \bot, \bot) \leftarrow bag\_permutation(s, v_1, v_2, v_3, v_4).$
$bag\_subset(s, \bot, \bot, \bot, \bot).$

Fig. 9. Further auxiliary predicates.

$\hat{G}$, and $\hat{B}$. Initially the sets $\hat{R}$, $\hat{G}$, and $\hat{B}$ are empty and we start the recursion by choosing s as the root node of the tree decomposition $\mathcal{T}$. As (R, G, B) we choose any combination, such that the fact $solve(s, R, G, B)$ is true in the minimal model of the program. Then we traverse the tree decomposition $\mathcal{T}$ in top-down direction by carrying out the following action at every node s:

(1) If s is a leaf node, then we set $\hat{R} := \hat{R} \cup R$, $\hat{G} := \hat{G} \cup G$, and $\hat{B} := \hat{B} \cup B$.

(2) Suppose that s is an element introduction node with child node s_1 and newly introduced vertex v. Then, depending on whether v is in R, G, or B, we set $\hat{R} := \hat{R} \cup \{v\}$, $\hat{G} := \hat{G} \cup \{v\}$, or $\hat{B} := \hat{B} \cup \{v\}$, respectively. The recursion continues with the values $(s_1, R \setminus \{v\}, G \setminus \{v\}, B \setminus \{v\})$.

(3) Suppose that s is an element removal node with child node s_1 and removed vertex v. By construction, there exists a fact $solve(s, R, G, B)$ which is true in the minimal model of the program. Hence, at least one of the following facts is also true in the minimal model of the program: $solve(s_1, R \cup \{v\}, G, B)$, $solve(s_1, R, G \cup \{v\}, B)$, and $solve(s_1, R, G, B \cup \{v\})$. We thus continue the recursion with the value combination $(s_1, R \cup \{v\}, G, B)$, respectively, $(s_1, R, G \cup \{v\}, B)$, respectively, $(s_1, R, G, B \cup \{v\})$.

(4) If s is a branch node with child nodes s_1 and s_2, then we continue with two calls to our recursive procedure: one with the value combination (s_1, R, G, B) and one with (s_2, R, G, B).

5.2 The Primality Decision Problem

Recall from Section 2.2 that we represent a relational schema (R, F) as a τ-structure with $\tau = \{fd, att, lh, rh\}$. Moreover, recall that, in Section 5, we consider normalized tree decompositions with 3 kinds of internal nodes, namely element removal nodes, element introduction nodes, and branch nodes

(cf. Definition 2.6). With our representation of relational schemas (R, F) as structures, the domain elements are the attributes and FDs in (R, F). Hence, we distinguish two kinds of element removal nodes, namely, *attribute removal nodes* and *FD removal nodes*. Likewise, we have two kinds of element introduction nodes, namely, *attribute introduction nodes* and *FD introduction nodes*. Moreover, it is convenient to denote the bags as a *pairs of sets* (At, Fd), where At is a set of attributes and Fd is a set of FDs. Finally, it will greatly simplify the presentation of our datalog program if we require that, whenever an FD $f \in F$ is contained in a bag of the tree decomposition, then the attribute $rhs(f)$ must be as well. In the worst case, this may double the width of the resulting decomposition.

Suppose that a schema (R, F) together with a tree decomposition $\mathcal{T}$ of width w is given as a τ_{td}-structure with $\tau_{td} = \{fd, att, lh, rh, root, leaf, child_1, child_2, bag\}$. In Figure 10, we describe a datalog program, where the input is given as an attribute $a \in R$ and a τ_{td}-structure, such that a occurs in the bag at the root of the tree decomposition.

Analogously to Section 5.1, we are using lower case letters s, f, and b (possibly with subscripts) as datalog variables for a single node in $\mathcal{T}$, for a single FD, or for a single attribute in R, respectively. Uppercase letters are used as datalog variables denoting sets of attributes (in the case of $Y, At, C^o, \Delta C$) or sets of FDs (in the case of Fd, FY, FC). In addition, C^o is considered as an ordered set (indicated by the superscript o). When we write $C^o \uplus \{b\}$, we mean that b is arbitrarily "inserted" into C^o, leaving the order of the remaining elements unchanged. Again, the cardinality of these (ordered) sets is restricted by the size $w + 1$ of the bags, where w is a fixed constant. In addition to $\uplus$ (disjoint union) we are now also using the set operators $\cup$, $\cap$, $\subseteq$, and $\in$. For the fixed-size (ordered) sets under consideration here, one could, of course, easily replace these operators by pure datalog expressions. In Figures 7, 8, and 9 we have already seen how set variables and some set operators can be realized in pure datalog for treewidth $w = 3$. In Figures 11 and 12, the realization of further set operators is presented.

For the input schema (R, F) with tree decomposition $\mathcal{T}$, we use the following notation: we write $FD(s)$ to denote the FDs in the bag of s and $FD(\mathcal{T}_s)$ to denote the FDs that occur in any bag in $\mathcal{T}_s$. Analogously, we write $Att(s)$ and $Att(\mathcal{T}_s)$ as a short-hand for the attributes occurring in the bag of s, respectively, in any bag in $\mathcal{T}_s$. Our PRIMALITY program in Figure 10 checks the primality of an attribute a via the criterion used for the MSO characterization in Example 2.7: we have to search for an attribute set $\mathcal{Y} \subseteq R$, such that $\mathcal{Y}$ is closed with respect to F (i.e., $\mathcal{Y}^+ = \mathcal{Y}$), $a \notin \mathcal{Y}$, and $(\mathcal{Y} \cup \{a\})^+ = R$, that is, $\mathcal{Y} \cup \{a\}$ is a superkey but $\mathcal{Y}$ is not.

At the heart of our PRIMALITY-program is the intensional predicate $solve(s, Y, FY, C^o, \Delta C, FC)$ with the following intended meaning: s denotes a node in $\mathcal{T}$. Y (respectively, C^o) is the intersection of $\mathcal{Y}$ (respectively, of $R \setminus \mathcal{Y}$) with $Att(s)$. We consider $R \setminus \mathcal{Y}$ as ordered with respect to an appropriate derivation sequence of R from $\mathcal{Y} \cup \{a\}$, that is, suppose that $\mathcal{Y} \cup \{A_0\} \to \mathcal{Y} \cup \{A_0, A_1\} \to \mathcal{Y} \cup \{A_0, A_1, A_2\} \to \cdots \to \mathcal{Y} \cup \{A_0, A_1, \ldots, A_n\}$, such that $A_0 = a$

Program PRIMALITY

/* leaf node. */
$solve(s, Y, FY, C^o, \Delta C, FC) \leftarrow leaf(s)$, $bag(s, At, Fd)$, $Y \cup C^o = At$, $Y \cap C^o = \emptyset$, $outside(FY, Y, At, Fd)$, $FC \subseteq Fd$, $consistent(FC, C^o)$, $\Delta C = \{rhs(f) \mid f \in FC\}$, $\Delta C \subseteq C^o$.

/* attribute introduction node. */
$solve(s, Y \uplus \{b\}, FY, C^o, \Delta C, FC) \leftarrow bag(s, At \uplus \{b\}, Fd)$, $child_1(s_1, s)$, $bag(s_1, At, Fd)$, $solve(s_1, Y, FY, C^o, \Delta C, FC)$.
$solve(s, Y, FY, C^o \uplus \{b\}, \Delta C, FC) \leftarrow bag(s, At \uplus \{b\}, Fd)$, $child_1(s_1, s)$, $bag(s_1, At, Fd)$, $consistent(FC, C^o \uplus \{b\})$, $solve(s_1, Y, FY_1, C^o, \Delta C, FC)$, $outside(FY_2, Y, At, Fd)$, $FY = FY_1 \cup FY_2$.

/* FD introduction node. */
$solve(s, Y, FY, C^o, \Delta C, FC) \leftarrow bag(s, At, Fd \uplus \{f\})$, $child_1(s_1, s)$, $bag(s_1, At, Fd)$, $rh(b, f)$, $b \in Y$, $solve(s_1, Y, FY, C^o, \Delta C, FC)$.
$solve(s, Y, FY, C^o, \Delta C \uplus \{b\}, FC \uplus \{f\}) \leftarrow bag(s, At, Fd \uplus \{f\})$, $child_1(s_1, s)$, $bag(s_1, At, Fd)$, $rh(b, f)$, $b \in C^o$, $solve(s_1, Y, FY_1, C^o, \Delta C, FC)$, $consistent(\{f\}, C^o)$, $outside(FY_2, Y, At, \{f\})$, $FY = FY_1 \cup FY_2$.
$solve(s, Y, FY, C^o, \Delta C, FC) \leftarrow bag(s, At, Fd \uplus \{f\})$, $child_1(s_1, s)$, $bag(s_1, At, Fd)$, $rh(b, f)$, $b \in C^o$, $solve(s_1, Y, FY_1, C^o, \Delta C, FC)$, $outside(FY_2, Y, At, \{f\})$, $FY = FY_1 \cup FY_2$.

/* attribute removal node. */
$solve(s, Y, FY, C^o, \Delta C, FC) \leftarrow bag(s, At, Fd)$, $child_1(s_1, s)$, $bag(s_1, At \uplus \{b\}, Fd)$, $solve(s_1, Y \uplus \{b\}, FY, C^o, \Delta C, FC)$.
$solve(s, Y, FY, C^o, \Delta C, FC) \leftarrow bag(s, At, Fd)$, $child_1(s_1, s)$, $bag(s_1, At \uplus \{b\}, Fd)$, $solve(s_1, Y, FY, C^o \uplus \{b\}, \Delta C \uplus \{b\}, FC)$.

/* FD removal node. */
$solve(s, Y, FY, C^o, \Delta C, FC) \leftarrow bag(s, At, Fd)$, $child_1(s_1, s)$, $bag(s_1, At, Fd \uplus \{f\})$, $rh(b, f)$, $b \in Y$, $solve(s_1, Y, FY, C^o, \Delta C, FC)$.
$solve(s, Y, FY, C^o, \Delta C, FC) \leftarrow bag(s, At, Fd)$, $child_1(s_1, s)$, $bag(s_1, At, Fd \uplus \{f\})$, $rh(b, f)$, $b \in C^o$, $solve(s_1, Y, FY \uplus \{f\}, C^o, \Delta C, FC \uplus \{f\})$.
$solve(s, Y, FY, C^o, \Delta C, FC) \leftarrow bag(s, At, Fd)$, $child_1(s_1, s)$, $bag(s_1, At, Fd \uplus \{f\})$, $rh(b, f)$, $b \in C^o$, $solve(s_1, Y, FY \uplus \{f\}, C^o, \Delta C, FC)$, $f \notin FC$.

/* branch node. */
$solve(s, Y, FY_1 \cup FY_2, C^o, \Delta C_1 \cup \Delta C_2, FC) \leftarrow bag(s, At, Fd)$, $child_1(s_1, s)$, $bag(s_1, At, Fd)$, $child_2(s_2, s)$, $bag(s_2, At, Fd)$, $solve(s_1, Y, FY_1, C^o, \Delta C_1, FC)$, $solve(s_2, Y, FY_2, C^o, \Delta C_2, FC)$, $unique(\Delta C_1, \Delta C_2, FC)$.

/* result (at the root node). */
$success \leftarrow root(s)$, $bag(s, At, Fd)$, $a \in At$, $solve(s, Y, FY, C^o, \Delta C, FC)$, $a \notin Y$, $FY = \{f \in Fd \mid rhs(f) \notin Y\}$, $\Delta C = C^o \setminus \{a\}$.

Fig. 10. Primality test.

and $\mathcal{Y} \cup \{A_0, A_1, \ldots, A_n\} = R$. Without loss of generality, the A_i's may be assumed to be pairwise distinct. Then for any two $i \neq j$, we simply set $A_i < A_j$ if and only if $i < j$. By the connectedness condition on $\mathcal{T}$, our datalog program ensures that the order on each subset C^o of $R \setminus \mathcal{Y}$ is consistent with the overall ordering.

The argument FY of the *solve* predicate is used to guarantee that $\mathcal{Y}$ is indeed closed. Informally, FY contains those FDs in $FD(s)$ for which we have

Auxiliary Predicates: Set Operations for Treewidth = 3, Part 1.

/* $X_1 \subseteq X_2$ */
$subset(s, v_1, v_2, v_3, v_4, v_1, v_2, v_3, v_4) \leftarrow bag\_subset(s, v_1, v_2, v_3, v_4).$
$subset(s, v_1, v_2, v_3, \bot, v_1, v_2, v_3, v_4) \leftarrow bag\_subset(s, v_1, v_2, v_3, v_4).$
$subset(s, v_1, v_2, \bot, \bot, v_1, v_2, v_3, v_4) \leftarrow bag\_subset(s, v_1, v_2, v_3, v_4).$
$subset(s, v_1, \bot, \bot, \bot, v_1, v_2, v_3, v_4) \leftarrow bag\_subset(s, v_1, v_2, v_3, v_4).$
$subset(s, \bot, \bot, \bot, \bot, v_1, v_2, v_3, v_4) \leftarrow bag\_subset(s, v_1, v_2, v_3, v_4).$

/* $v \in X$ */
$element(s, v_1, v_1, v_2, v_3, v_4) \leftarrow bag\_subset(s, v_1, v_2, v_3, v_4), v_1 \neq \bot.$
$element(s, v_2, v_1, v_2, v_3, v_4) \leftarrow bag\_subset(s, v_1, v_2, v_3, v_4), v_2 \neq \bot.$
$element(s, v_3, v_1, v_2, v_3, v_4) \leftarrow bag\_subset(s, v_1, v_2, v_3, v_4), v_3 \neq \bot.$
$element(s, v_4, v_1, v_2, v_3, v_4) \leftarrow bag\_subset(s, v_1, v_2, v_3, v_4), v_4 \neq \bot.$

/* $X_1 \cap X_2 = X$ */
$intersection(s, v_1, v_2, v_3, v_4, v_1, v_2, v_3, v_4, v_1, v_2, v_3, v_4) \leftarrow bag\_subset(s, v_1, v_2, v_3, v_4).$
$intersection(s, v_1, v_2, v_3, v_4, v_1, v_2, v_3, u_1, v_1, v_2, v_3, \bot) \leftarrow$
$\quad bag\_subset(s, v_1, v_2, v_3, v_4), bag\_subset(s, v_1, v_2, v_3, u_1), u_1 \neq v_4.$
$intersection(s, v_1, v_2, v_3, v_4, v_1, v_2, u_1, u_2, v_1, v_2, \bot, \bot) \leftarrow$
$\quad bag\_subset(s, v_1, v_2, v_3, v_4), bag\_subset(s, v_1, v_2, u_1, u_2), u_1 \neq v_3, u_1 \neq v_4, u_2 \neq v_3, u_2 \neq v_4.$
$intersection(s, v_1, v_2, v_3, v_4, v_1, u_1, u_2, u_3, v_1, \bot, \bot, \bot) \leftarrow bag\_subset(s, v_1, v_2, v_3, v_4),$
$\quad bag\_subset(s, v_1, u_1, u_2, u_3), u_1 \neq v_2, u_1 \neq v_3, u_1 \neq v_4, \ldots, u_3 \neq v_2, u_3 \neq v_3, u_3 \neq v_4.$
$intersection(s, v_1, v_2, v_3, v_4, u_1, u_2, u_3, u_4, \bot, , \bot, \bot, \bot) \leftarrow bag\_subset(s, v_1, v_2, v_3, v_4),$
$\quad bag\_subset(s, u_1, u_2, u_3, u_4), u_1 \neq v_1, u_1 \neq v_2, u_1 \neq v_3, u_1 \neq v_4, \ldots, u_4 \neq v_1, \ldots, u_4 \neq v_4.$
$intersection(s, v_1, v_2, v_3, \bot, v_1, v_2, v_3, u_1, v_1, v_2, v_3, \bot) \leftarrow$
$\quad bag\_subset(s, v_1, v_2, v_3, \bot), bag\_subset(s, v_1, v_2, v_3, u_1).$
$intersection(s, v_1, v_2, v_3, \bot, v_1, v_2, u_1, u_2, v_1, v_2, \bot, \bot) \leftarrow$
$\quad bag\_subset(s, v_1, v_2, v_3, \bot), bag\_subset(s, v_1, v_2, u_1, u_2), u_1 \neq v_3, u_2 \neq v_3.$
$intersection(s, v_1, v_2, v_3, \bot, v_1, u_1, u_2, u_3, v_1, \bot, \bot, \bot) \leftarrow bag\_subset(s, v_1, v_2, v_3, \bot),$
$\quad bag\_subset(s, v_1, u_1, u_2, u_3), u_1 \neq v_2, u_1 \neq v_3, \ldots, u_3 \neq v_2, u_3 \neq v_3,$
$intersection(s, v_1, v_2, v_3, \bot, u_1, u_2, u_3, u_4, \bot, , \bot, \bot, \bot) \leftarrow bag\_subset(s, v_1, v_2, v_3, \bot),$
$\quad bag\_subset(s, u_1, u_2, u_3, u_4), u_1 \neq v_1, u_1 \neq v_2, u_1 \neq v_3, \ldots, u_4 \neq v_1, u_4 \neq v_2, u_4 \neq v_3.$
$intersection(s, v_1, \bot, \bot, \bot, v_1, u_1, u_2, u_3, v_1, \bot, \bot, \bot) \leftarrow$
$\quad bag\_subset(s, v_1, \bot, \bot, \bot), bag\_subset(s, v_1, u_1, u_2, u_3).$
$intersection(s, \bot, \bot, \bot, \bot, u_1, u_2, u_3, u_4, \bot, \bot, \bot, \bot) \leftarrow bag\_subset(s, u_1, u_2, u_3, u_4).$

Fig. 11. Datalog instantiation of set operators, Part 1.

already verified (on the bottom-up traversal of the tree decomposition) that they do not constitute a contradiction with the closedness of $\mathcal{Y}$. In other words, either $rhs(f) \in \mathcal{Y}$ or there exists an attribute in $lhs(f) \cap At(\mathcal{T}_s)$ which is not in $\mathcal{Y}$.

The arguments ΔC and FC of the *solve* predicate are used to ensure that $(\mathcal{Y} \cup \{a\})^+ = R$ indeed holds: the intended meaning of the set FC is that it contains those FDs in $FD(s)$ which are used in the above derivation sequence. Moreover, ΔC contains those attributes from $Att(s)$ for which we have already shown that they can be derived from $\mathcal{Y}$ and smaller atoms in C^o.

More precisely, for all values $s, Y, FY, C^o, \Delta C, FC$, the ground fact $solve(s, Y, FY, C^o, \Delta C, FC)$ shall be true in the minimal model of the program and the input structure if and only if the following condition holds:

PROPERTY B. There exist extensions $\hat{Y}$ of Y and $\hat{C}^o$ of C^o to $Att(\mathcal{T}_s)$ and an extension $\hat{FC}$ of FC to $FD(\mathcal{T}_s)$, such that

Auxiliary Predicates: Set Operations for Treewidth = 3, Part 2.

/* $X_1 \uplus X_2 = X$ */

$disjoint(s, v_1, v_2, v_3, v_4, \bot, \bot, \bot, \bot, v_1, v_2, v_3, v_4) \leftarrow bag\_subset(s, v_1, v_2, v_3, v_4).$

$disjoint(s, v_1, v_2, v_3, \bot, v_4, \bot, \bot, \bot, v_1, v_2, v_3, v_4) \leftarrow$
$\quad bag\_subset(s, v_1, v_2, v_3, \bot), bag\_subset(s, v_4, \bot, \bot, \bot), v_1 \neq v_4, v_2 \neq v_4, v_3 \neq v_4.$

$disjoint(s, v_1, v_2, \bot, \bot, v_3, v_4, \bot, \bot, v_1, v_2, v_3, v_4) \leftarrow bag\_subset(s, v_1, v_2, \bot, \bot),$
$\quad bag\_subset(s, v_3, v_4, \bot, \bot), v_1 \neq v_3, v_1 \neq v_4, v_2 \neq v_3, v_2 \neq v_4.$

$disjoint(s, v_1, \bot, \bot, \bot, v_2, v_3, v_4, \bot, v_1, v_2, v_3, v_4) \leftarrow$
$\quad bag\_subset(s, v_1, \bot, \bot, \bot), bag\_subset(s, v_2, v_3, v_4, \bot), v_1 \neq v_2, v_1 \neq v_3, v_1 \neq v_4.$

$disjoint(s, \bot, \bot, \bot, \bot, v_1, v_2, v_3, v_4, v_1, v_2, v_3, v_4) \leftarrow bag\_subset(s, v_1, v_2, v_3, v_4).$

$disjoint(s, v_1, v_2, v_3, \bot, \bot, \bot, \bot, \bot, v_1, v_2, v_3, \bot) \leftarrow bag\_subset(s, v_1, v_2, v_3, \bot).$

$disjoint(s, v_1, v_2, \bot, \bot, v_3, \bot, \bot, \bot, v_1, v_2, v_3, \bot) \leftarrow$
$\quad bag\_subset(s, v_1, v_2, \bot, \bot), bag\_subset(s, v_3, \bot, \bot, \bot), v_1 \neq v_3, v_2 \neq v_3.$

$disjoint(s, v_1, \bot, \bot, \bot, v_2, v_3, \bot, \bot, v_1, v_2, v_3, \bot) \leftarrow$
$\quad bag\_subset(s, v_1, \bot, \bot, \bot), bag\_subset(s, v_2, v_3, \bot, \bot), v_1 \neq v_2, v_1 \neq v_3.$

$disjoint(s, \bot, \bot, \bot, \bot, v_1, v_2, v_3, \bot, v_1, v_2, v_3, \bot) \leftarrow bag\_subset(s, v_1, v_2, v_3, \bot).$

$\vdots$

$disjoint(s, \bot, \bot, \bot, \bot, \bot, \bot, \bot, \bot, \bot, \bot, \bot, \bot).$

/* $X_1 \cup X_2 = X$ */

$union(s, v_1, v_2, v_3, v_4, v_1, v_2, v_3, v_4, v_1, v_2, v_3, v_4) \leftarrow bag\_subset(s, v_1, v_2, v_3, v_4).$

$union(s, v_1, v_2, v_3, v_4, v_1, v_2, v_3, \bot, v_1, v_2, v_3, v_4) \leftarrow$
$\quad bag\_subset(s, v_1, v_2, v_3, v_4), bag\_subset(s, v_1, v_2, v_3, \bot).$

$union(s, v_1, v_2, v_3, \bot, v_1, v_2, v_3, v_4, v_1, v_2, v_3, v_4) \leftarrow$
$\quad bag\_subset(s, v_1, v_2, v_3, \bot), bag\_subset(s, v_1, v_2, v_3, v_4).$

$union(s, v_1, v_2, v_3, \bot, v_1, v_2, v_4, \bot, v_1, v_2, v_3, v_4) \leftarrow$
$\quad bag\_subset(s, v_1, v_2, v_3, \bot), bag\_subset(s, v_1, v_2, v_4, \bot), v_3 \neq v_4.$

$union(s, v_1, v_2, v_3, \bot, v_1, v_2, \bot, \bot, v_1, v_2, v_3, \bot) \leftarrow$
$\quad bag\_subset(s, v_1, v_2, v_3, \bot), bag\_subset(s, v_1, v_2, \bot, \bot).$

$union(s, v_1, v_2, \bot, \bot, v_1, v_2, v_3, \bot, v_1, v_2, v_3, \bot) \leftarrow$
$\quad bag\_subset(s, v_1, v_2, \bot, \bot), bag\_subset(s, v_1, v_2, v_3, \bot).$

$union(s, v_1, v_2, \bot, \bot, v_1, v_2, \bot, \bot, v_1, v_2, \bot, \bot) \leftarrow bag\_subset(s, v_1, v_2, \bot, \bot).$

$union(s, v_1, v_2, v_3, \bot, v_1, v_4, \bot, \bot, v_1, v_2, v_3, v_4) \leftarrow$
$\quad bag\_subset(s, v_1, v_2, v_3, \bot), bag\_subset(s, v_1, v_4, \bot, \bot), v_2 \neq v_4, v_3 \neq v_4.$

$union(s, v_1, v_2, \bot, \bot, v_1, v_3, v_4, \bot, v_1, v_2, v_3, v_4) \leftarrow$
$\quad bag\_subset(s, v_1, v_2, \bot, \bot), bag\_subset(s, v_1, v_3, v_4, \bot), v_2 \neq v_3, v_2 \neq v_4.$

$\vdots$

$union(s, \bot, \bot, \bot, \bot, \bot, \bot, \bot, \bot, \bot, \bot, \bot, \bot).$

Fig. 12. Datalog instantiation of set operators, Part 2.

(1) $\hat{Y}$ and $\hat{C}^o$ form a partition of $Att(\mathcal{T}_s)$;
(2) for every $f \in FD(\mathcal{T}_s) \setminus FD(s)$, if $rhs(f) \notin \hat{Y}$, then $lhs(f) \not\subseteq \hat{Y}$; moreover, $FY = \{f \in FD(s) \mid rhs(f) \notin \hat{Y}$ and $lhs(f) \cap Att(\mathcal{T}_s) \not\subseteq \hat{Y}\}$;
(3) For every $f \in \hat{FC}$, f is consistent with the order on $\hat{C}^o$, that is, for every $f \in \hat{FC}$, $rhs(f) \in \hat{C}^o$ holds and, for every $b \in lhs(f) \cap \hat{C}^o$, $b < rhs(f)$ holds;
(4) $\Delta C \cup \hat{C}^o \setminus Att(s) = \{rhs(f) \mid f \in \hat{FC}\}$.

The main task of the program is the computation of all facts $solve(s, Y, FY, C^o, \Delta C, FC)$ by means of a bottom-up traversal of the tree decomposition. The other predicates have the following meaning:

—$outside(FY, Y, At, Fd)$ is true in the minimal model if and only if $FY = \{f \in Fd \mid rhs(f) \notin Y$ and $lhs(f) \cap At \not\subseteq Y\}$, that is, for every $f \in FY$, $rhs(f)$ is outside Y but this will never conflict with the closedness of Y because $lhs(f)$ contains an attribute from outside Y.

—$consistent(FC, C^o)$ is true in the minimal model if and only if $\forall f \in FC$ we have $rhs(f) \in C^o$ and $\forall b \in lhs(f) \cap C^o$: $b < rhs(f)$, that is, the FDs in FC are only used to derive greater attributes from smaller ones (and attributes from $\mathcal{Y}$).

—The fact $unique(\Delta C_1, \Delta C_2, FC)$ is true in the minimal model if and only if the condition $\Delta C_1 \cap \Delta C_2 = \{b \mid b = rhs(f)$ for some $f \in FC\}$ holds. The *unique* predicate is only used in the body of the rule for branch nodes. Its purpose is to avoid that an attribute in $R \setminus \mathcal{Y}$ is derived via two different FDs in the two subtrees at the child nodes of the branch node.

—The 0-ary predicate *success* indicates if the fixed attribute a is prime in the schema encoded by the input structure.

The PRIMALITY-program has the following properties.

LEMMA 5.2. *The solve predicate has the intended meaning described above, that is, for all values s, Y, FY, C^o, ΔC, FC, the ground fact $solve(s, Y, FY, C^o, \Delta C, FC)$ is true in the minimal model of the PRIMALITY program and the input structure if and only if Property B holds.*

PROOF SKETCH. The lemma can be shown by structural induction on $\mathcal{T}$. We restrict ourselves here to outlining the ideas underlying the various rules of the PRIMALITY-program. The induction itself is then obvious and therefore omitted.

(1) *Leaf nodes.* The rule for a leaf node s realizes two "guesses," so to speak: (i) a partition of $At(s)$ into Y and C^o together with an ordering on C^o and (ii) the subset $FC \subseteq Fd(s)$ of FDs which are used in the derivation sequence of $R \setminus \mathcal{Y}$ from $\mathcal{Y} \cup \{a\}$. The remaining variables are thus fully determined: FY is determined via the *outside*predicate, while ΔC is determined via the equality $\Delta C = \{rhs(f) \mid f \in FC\}$. Finally, the body of the rule contains the checks $consistent(FC, C^o)$ and $\Delta C \subseteq C^o$ to make sure that (at least at the leaf node s) the "guesses" are allowed.

(2) *Attribute introduction node.* The two rules are used to distinguish two cases whether the new attribute b is added to Y or to C^o. If b is added to Y then all arguments of the *solve*fact at the child node s_1 of s remain unchanged at s. In contrast, if b is inserted into C^o then the following actions are required.

The atom $consistent(FC, C^o \uplus \{b\})$ makes sure that the rules in FC are consistent with the ordering of C^o, that is, it must not happen that the new attribute b occurs in $lhs(f)$ for some $f \in FC$, such that $b > rhs(f)$ holds.

The new attribute b outside Y may possibly allow us to verify for some additional FDs that they do not contradict the closedness of $\mathcal{Y}$. The atom $outside(FY_2, Y, At, Fd)$ determines the set FY_2 which contains all FDs with $rhs(f) \notin Y$ but with some attribute from C^o (in particular, the new attribute b) in $lhs(f)$.

Recall that we are requiring that, whenever an FD $f \in F$ is contained in a bag of the tree decomposition, then the attribute $rhs(f)$ is as well. Hence, since the attribute b has just been introduced on our bottom-up traversal of the tree decomposition, we can be sure that b does not occur on the right-hand side of any FD in the bag of s. Thus, ΔC is not affected by the transition from s_1 to s.

(3) *FD introduction node.* The three rules distinguish, in total, three cases: first, does $rhs(f) \in Y$ or $rhs(f) \in C^o$ hold? (Recall that we assume that every bag containing some FD also contains the right-hand side of this FD.) The latter case is then further divided into the subcases if f is used for the derivation of $R \setminus \mathcal{Y}$ or not. The first rule deals with the case $rhs(f) \in Y$. Then all arguments of the *solve* fact at the child node s_1 of s remain unchanged at s.

The second rule addresses the case that $rhs(f) \in C^o$ and f is used for the derivation of $R \setminus \mathcal{Y}$. Then the attribute $rhs(f)$ is added to ΔC. The disjoint union makes sure that this attribute has not yet been derived by another rule with the same right-hand side. The atom $consistent(FC, C^o \uplus \{b\})$ is used to check the consistency of f with the ordering of C^o. The atom $outside(FY_2, Y, At, Fd)$ is used to check if f may be added to FY, that is, if some attribute in $lhs(f)$ is in C^o.

The third rule refers to the case that $rhs(f) \in C^o$ and f is *not* used for the derivation of $R \setminus \mathcal{Y}$. Again, the atom $outside(FY_2, Y, At, Fd)$ is used to check if f may be added to FY.

(4) *Attribute removal node.* The two rules are used to distinguish two cases whether the attribute b was in Y or in C^o. If b was in Y then all arguments of the *solve* fact at the child node s_1 of s remain unchanged at s. In contrast, if b was in C^o, then we have to check (by pattern matching with the fact $solve(s_1, \ldots, \Delta C \uplus \{b\}, \ldots)$) that a rule f for deriving b has already been found. Recall that, on our bottom-up traversal of $\mathcal{T}$, when we first encounter an attribute b, it is either added to Y or C^o. If b is added to C^o then we eventually have to determine the FD by which b is derived. Hence, initially, b is in C^o but not in ΔC. However, when b is finally removed from the bag then its derivation must have been verified. The arguments Y, FY, and FC are of course not affected by this attribute removal.

(5) *FD removal node.* Similarly to the FD introduction node, we distinguish, in total, three cases. If $rhs(f) \in Y$ then all arguments of the *solve* fact at the child node s_1 of s remain unchanged at s. If $rhs(f) \in C^o$ then we further distinguish the subcases if f is used for the derivation of $R \setminus \mathcal{Y}$ or not. The second and third rule refer to these two subcases. The action carried out by these two rules is the same, namely, it has to be checked (by pattern matching with the fact $solve(s_1, \ldots, FY \uplus \{f\}, \ldots)$) that f does not constitute

a contradiction with the closedness of $\mathcal{Y}$. In other words, since $rhs(f) \in C^o$, we must have encountered (on our bottom-up traversal of $\mathcal{T}$) an attribute in $lhs(f)$ which is outside $\mathcal{Y}$.

(6) *Branch node.* Recall that a branch node s and its two child nodes s_1 and s_2 have identical bags by our notion of normalized tree decompositions. The argument of the *solve* fact at s is then determined from the arguments at s_1 and s_2 as follows: the arguments Y and C^o must have the same value at all three nodes s, s_1, and s_2. Likewise, FC (containing the FDs from the bags at these nodes which are used in the derivation of $R \setminus \mathcal{Y}$) must be identical. In contrast, FY and ΔC are obtained as the union of the corresponding arguments in the *solve*-facts at the child nodes s_1 and s_2, that is, it suffices to verify at one of the child nodes s_1 or s_2 that some FD does not contradict the closedness of Y and that some attribute in C^o is derived by some FD.

Recall that we define an order on the attributes in $R \setminus \mathcal{Y}$ by means of some derivation sequence of $R \setminus \mathcal{Y}$ from $\mathcal{Y} \cup \{a\}$. Hence, we have to make sure that every attribute in $R \setminus \mathcal{Y}$ is derived only once in this derivation sequence. In other words, for every $b \in R \setminus (\mathcal{Y} \cup \{a\})$, we use exactly one FD f with $rhs(f) = b$ in our derivation sequence. The atom $unique(\Delta C_1, \Delta C_2, FC)$ in the rule body ensures that no attribute in $R \setminus \mathcal{Y}$ is derived via two different FDs in the two subtrees at the child nodes of the branch node.

THEOREM 5.3. *The datalog program in Figure* 10 *decides the PRIMALITY problem for a fixed attribute a, that is, the fact "success" is true in the minimal model of this program and the input τ_{td}-structure $\mathcal{A}_{td}$ if and only if $\mathcal{A}_{td}$ encodes a relational schema (R, F) together with a tree decomposition $\mathcal{T}$ of (R, F), such that a is part of a key. Moreover, for any schema (R, F) and tree decomposition $\mathcal{T}$ of width at most w, the program can be evaluated in time $\mathcal{O}(4^{(w \log w)} * |(R, F)|)$.*

PROOF. By Lemma 5.2, the predicate *solve* indeed has the meaning according to Property B. Thus, the rule with head *success* reads as follows: *success* is true in the minimal model if and only if s denotes the root of $\mathcal{T}$, a is an attribute in the bag at s, and Y is the intersection of the desired attribute set $\mathcal{Y}$ with $Att(s)$, that is, (1) $\mathcal{Y}$ is closed (this is ensured by the condition that $\{f \in Fd \mid rhs(f) \notin Y\} = FY$), (2) $a \notin \mathcal{Y}$ and, finally, (3) all attributes in $R \setminus (\mathcal{Y} \cup \{a\})$ are indeed determined by $\mathcal{Y} \cup \{a\}$ (this is ensured by the condition $\Delta C = C^o \setminus \{a\}$).

The upper bound on the time complexity is shown by a similar argument as in the proof of Theorem 5.1: the datalog program in Figure 10 is equivalent to a ground program $\mathcal{P}'$ where each rule of $\mathcal{P}$ is replaced by $\mathcal{O}(4^{(w \log w)} * |(R, F)|)$ ground rules. A detailed proof of this fact can be given by a case distinction over the possible kinds of nodes in the tree decomposition. In all cases, the crucial observation is that the number of possible instantiations of the arguments of the $solve(s, Y, FY, C^o, \Delta C, FC)$ predicate is bounded as follows: suppose that the bag at node s consists of k attributes and l FDs with $k + l \leq w + 1$. The arguments Y and C^o are disjoint subsets of $Att(s)$; moreover, C^o is ordered. Hence, there are at most $2^k * k!$ possible instantiations of these two arguments.

The argument FC is a subset of $FD(s)$. Hence, there are at most 2^l possible instantiations of this argument. Finally, ΔC is fully determined by the choice of C^o and FC; likewise, FY is fully determined by the choice of Y. In total, the number of ground instantiations of each rule in $\mathcal{P}$ is bounded by $(w+1)! * (w+1)! = \mathcal{O}(2^{(w \log w)} * 2^{(w \log w)}) = \mathcal{O}(4^{(w \log w)})$ for every node s in the tree decomposition $\mathcal{T}$ of $|(R, F)|$. Since the size of $\mathcal{T}$ is linearly bounded in $|(R, F)|$, we get the upper bound $\mathcal{O}(4^{(w \log w)} * |(R, F)|)$ on the size of $\mathcal{P}'$. This ground program can then be evaluated in linear time. □

5.3 The Primality Enumeration Problem

In order to extend the PRIMALITY algorithm from the previous section to a monadic predicate selecting all prime attributes in a schema, a naive first attempt might look as follows: one can consider the tree decomposition $\mathcal{T}$ as rooted at various nodes, such that each $a \in R$ is contained in the bag of one such root node. Then, for each a and corresponding tree decomposition $\mathcal{T}$, we run the algorithm from Figure 10. Obviously, this method has *quadratic* time complexity with respect to the data size. However, in this section, we describe a *linear* time algorithm.

The idea of this algorithm is to implement a top-down traversal of the tree decomposition in addition to the bottom-up traversal realized by the program in Figure 10. For this purpose, we modify our notion of *normalized* tree decompositions in the following way: first, any tree decomposition can of course be transformed in such a way that every attribute $a \in R$ occurs in at least one leaf node of $\mathcal{T}$. Moreover, for every branch node s in the tree decomposition, we insert a new node u as new parent of s, such that u and s have identical bags. Hence, together with the two child nodes of s, each branch node is "surrounded" by three neighboring nodes with identical bags. It is thus guaranteed that a branch node always has two child nodes with identical bags, no matter where $\mathcal{T}$ is rooted. Moreover, this insertion of a new node also implies that the root node of $\mathcal{T}$ is not a branch node.

We propose the following algorithm for computing a monadic predicate *prime*(), which selects precisely the prime attributes in (R, F). In addition to the predicate *solve*, whose meaning was described by Property B in Section 5.2, we also compute a predicate $solve{\downarrow}$, whose meaning is described by replacing every occurrence of $\mathcal{T}_s$ in Property B by $\bar{\mathcal{T}}_s$. As the notation $solve{\downarrow}$ suggests, the computation of $solve{\downarrow}$ can be done via a top-down traversal of $\mathcal{T}$. Note that $solve{\downarrow}(s, \ldots)$ for a leaf node s of $\mathcal{T}$ is exactly the same as if we computed $solve(s, \ldots)$ for the tree rooted at s. Hence, we can define the predicate *prime*() as follows.

Program Monadic-Primality

$prime(a) \leftarrow leaf(s), bag(s, At, Fd), a \in At, solve{\downarrow}(s, Y, FY, C^o, \Delta C, FC), a \notin Y,$
$FY = \{f \in Fd \mid rhs(f) \notin Y\}, \Delta C = C^o \setminus \{a\}.$

By the intended meaning of $solve{\downarrow}$ and by the properties of the PRIMALITY algorithm in Section 5.2, we immediately get the following result. As far as the upper bound on the complexity is concerned, note that a ground program

$\mathcal{P}'$ equivalent to the program $\mathcal{P}$ for the PRIMALITY enumeration problem has essentially the double size of a ground program for the PRIMALITY decision problem. Hence, the upper bound on the complexity carries over from Theorem 5.3.

THEOREM 5.4. *The monadic predicate prime*() *as defined above selects precisely the prime attributes. Moreover, for any schema* (R, F) *and tree decomposition* $\mathcal{T}$ *of width at most* w*, the program can be evaluated in time* $\mathcal{O}(4^{(w \log w)} * |(R, F)|)$.

Note that, in all programs presented in Section 5, we consider the tree decomposition as part of the input. It has already been mentioned in Section 2.2 that, in theory, for every given value $w \geq 1$, it can be decided in linear time (with respect to the size of the input structure), if some structure has treewidth at most w. Moreover, in the case of a positive answer, a tree decomposition of width w can also be computed in linear time, (see Bodlaender [1996]). We have also mentioned in Section 2.2 that the practical usefulness of this linearity is limited due to excessively big constants [Koster et al. 2001]. At any rate, the improvement of tree decomposition algorithms is an area of very active research and considerable progress has recently been made in developing heuristic-based tree decomposition algorithms [Koster et al. 2001; Bodlaender and Koster 2006, 2008; van den Eijkhof et al. 2007].

6. IMPLEMENTATION AND RESULTS

To test our new datalog programs in terms of their scalability with a large number of attributes and rules, we have implemented the PRIMALITY program from Section 5.2 in C++. The experiments were conducted on Linux kernel 2.6.17 with an 1.60-GHz Intel Pentium(M) processor and 512 MB of memory. We measured the processing time of the PRIMALITY program on different input parameters such as the number of attributes and the number of FDs. The treewidth in all the test cases was 3. Note, however, that the applicability of our approach is not restricted to such a low treewidth. In fact, in Jakl et al. [2009], our approach has recently been adapted and applied to a Π_2^P-complete problem in the area of answer set programming. Even in this case, it was shown that our approach scales up to the treewidth of 7 and 1000 nodes in the tree decomposition.

6.1 Test Data Generation

Due to the lack of available test data, we generated a balanced normalized tree decomposition. Test data sets with increasing input parameters are then generated by expanding the tree in a depth-first style. We have ensured that all different kinds of nodes occur evenly in the tree decomposition.

6.2 Experimental Results

The outcome of the tests is shown in Table I, where *tw* stands for the treewidth, and #*Att*, #*FD*, and #*tn* stand for the number of attributes, FDs, and tree

Table I. Processing Time in milliseconds for PRIMALITY

tw	#Att	#FD	#tn	MD	MONA
3	3	1	3	0.1	650
3	6	2	12	0.2	9210
3	9	3	21	0.4	17930
3	12	4	34	0.5	—
3	21	7	69	0.8	—
3	33	11	105	1.0	—
3	45	15	141	1.2	—
3	57	19	193	1.6	—
3	69	23	229	1.8	—
3	81	27	265	1.9	—
3	93	31	301	2.2	—

nodes, respectively. The processing time (in milliseconds) obtained with our C++ implementation following the monadic datalog program in Section 5.2 are displayed in the column labeled MD. The measurements nicely reflect an essentially linear increase of the processing time with the size of the input. Moreover, there is obviously no big "hidden" constant which would render the linearity useless.

In Gottlob et al. [2006a], we proved the FPT of several nonmonotonic reasoning problems via Courcelle's Theorem. Moreover, we also carried out some experiments with a prototype implementation using MONA (see Klarlund et al. [2002]) for the MSO model checking. We have now extended these experiments with MONA to the PRIMALITY problem. The time measurements of these experiments are shown in the last column of Table I. Due to problems discussed in Gottlob et al. [2006a], MONA does not ensure linear data complexity. Hence, all tests below line 3 of the table failed with "out-of-memory" errors. Moreover, also in cases where the exponential data complexity does not yet "hurt," our datalog approach outperformsed the MSO-to-FTA approach by a factor of 1000 or even more.

6.3 Optimizations

In our implementation, we have realized several optimizations, which are highlighted below.

(1) *Succinct representation by nonmonadic datalog.* As was mentioned in the discussion following Theorem 5.1, our datalog programs can be regarded as succinct representations of big monadic datalog programs. Such a monadic datalog program has essentially the same size as the ground program obtained by computing all possible ground instantiations of the nonmonadic datalog program (like in the proofs of Theorems 5.1 and 5.3). However, the nonmonadic datalog program—combined with lazy grounding [Palù et al. 2009] (i.e., computing the required ground instances of a datalog rule only when the datalog rule is actually applied)—has the advantage that many possible instantiations may never have to be materialized since they are not "reachable" along the bottom-up computation. For instance, in the proof

of Theorem 5.1, the upper bound on the complexity was obtained by considering all possible ground instantiations of the rules in Figure 5. However, suppose that for some node s_1 in the tree decomposition, two vertices v_1, v_2 in the bag of s must always be assigned different colors, that is, in all $solve(s_1, R, G, B)$ facts derivable by the program, v_1 and v_2 are in different sets. Then the rules with head $solve(s, R', G', B')$ for the parent node s of s_1 will never need a ground instantiation with v_1 and v_2 jointly occurring in one of the sets R', G', or B'.

(2) *General optimizations and lazy grounding [Palù et al. 2009].* In principle, our implementation is based on the general idea of grounding followed by an evaluation of the ground program. This corresponds to the general technique to ensure linear time data complexity; cf. Theorem 4.5. As mentioned above, a further improvement is achieved by the natural idea of generating only those ground instances of rules which actually produce new facts.

(3) *Problem-specific optimizations of the nonmonadic datalog programs.* In the paragraph below Theorem 5.1, we have already mentioned that the datalog programs presented in Section 5 incorporate several problem-specific optimizations. The underlying idea of these optimizations is that many transitions which are kept track of by the generic construction in the proof of Theorem 4.6 (and, likewise, in the MSO-to-FTA approach) will not lead to a solution anyway. Hence, they are omitted in our datalog programs right from the beginning.

(4) *Language extensions.* As was mentioned in Section 5, we are using language constructs (in particular, for handling sets of attributes and FDs) which are not part of the datalog language (nevertheless they may be supported by datalog engines like the DLV-Complex extension of the DLV system [Leone et al. 2006]). In principle, they could be realized in datalog. Nevertheless, we preferred an efficient implementation of these constructs directly on C++ level. Further language extensions are conceivable and easy to realize.

(5) *Further improvements.* We are planning to implement further improvements. For instance, we are currently applying a strict bottom-up intuition as we compute new facts $solve(v, \ldots)$. However, some top-down guidance in the style of magic sets so as not to compute all possible such facts at each level would be desirable. Note that ultimately, at the root, only facts fulfilling certain conditions (like $a \notin Y$, etc.) are needed in case that an attribute a is indeed prime.

7. CONCLUSION

In this work, we have proposed a new approach based on monadic datalog to tackle a big class of fixed-parameter tractable problems. Theoretically, we have shown that every MSO-definable unary query over finite structures of bounded treewidth is also definable in monadic datalog. In fact, the resulting program even lies in a particularly efficient fragment of monadic datalog. Practically, we have put this approach to work by applying it to the 3-Colorability problem and the PRIMALITY problem in case of bounded treewidth. The experimental results thus obtained look very promising. They underline that datalog with

its potential for optimizations and its flexibility is clearly worth considering for this class of problems.

Recall that the PRIMALITY problem is closely related to an important problem in the area of artificial intelligence, namely the relevance problem of propositional abduction (i.e., given a system description in form of a propositional clausal theory and observed symptoms, one has to decide if some hypothesis is part of a possible explanation of the symptoms). Indeed, if the clausal theory is restricted to definite Horn clauses and if we are only interested in minimal explanations, then the relevance problem is basically the same as the problem of deciding primality in a subschema $R' \subseteq R$. Extending our *prime*() program (and, in particular, the *solve*() predicate) from Section 5 so as to test primality in a subschema is rather straightforward. On the other hand, extending such a program to abduction with arbitrary clausal theories (which is on the second level of the polynomial hierarchy; see Eiter and Gottlob [1995]) is much more involved. A monadic datalog program solving the relevance problem also in this general case was presented in Gottlob et al. [2008].

Our datalog program in Section 5 was obtained by an ad hoc construction rather than via a generic transformation from MSO. Nevertheless, we are convinced that the idea of a bottom-up propagation of certain conditions is quite generally applicable. We are therefore planning to tackle many more problems, whose FPT was established via Courcelle's Theorem, with this new approach. We have already incorporated some optimizations into our implementation. Further improvements are on the way (in particular, further heuristics to prune irrelevant parts of the search space).

ACKNOWLEDGMENTS

We are very grateful to the anonymous referees as well as to Michael Jakl, Stefan Rümmele, and Stefan Woltran for their valuable comments on previous versions of this article.

REFERENCES

ABITEBOUL, S., HULL, R., AND VIANU, V. 1995. *Foundations of Databases*. Addison-Wesley, Reading, MA.

ANDRÉKA, H., VAN BENTHEM, J., AND NÉMETI, I. 1995. Back and forth between modal logic and classical logic. *Logic J. IGPL 3*, 5, 685–720.

ARNBORG, S., LAGERGREN, J., AND SEESE, D. 1991. Easy problems for tree-decomposable graphs. *J. Algorithms. 12*, 2, 308–340.

BEERI, C. AND BERNSTEIN, P. A. 1979. Computational problems related to the design of normal form relational schemas. *ACM Trans. Datab. Syst. 4*, 1, 30–59.

BODLAENDER, H. L. 1996. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput. 25*, 6, 1305–1317.

BODLAENDER, H. L. AND KOSTER, A. M. C. A. 2006. Safe separators for treewidth. *Discrete Math. 306*, 3, 337–350.

BODLAENDER, H. L. AND KOSTER, A. M. C. A. 2008. Combinatorial optimization on graphs of bounded treewidth. *Comput. J. 51*, 3, 255–269.

CALÌ, A., GOTTLOB, G., AND LUKASIEWICZ, T. 2009a. Datalog$^{\pm}$: A unified approach to ontologies and integrity constraints. In *Proceedings of ICDT*. ACM International Conference Proceeding Series, vol. 361. ACM Press, New York, NY, 14–30.

CALÌ, A., GOTTLOB, G., AND LUKASIEWICZ, T. 2009b. A general datalog-based framework for tractable query answering over ontologies. In *Proceedings of PODS*. ACM Press, New York, NY, 77–86.

CERI, S., GOTTLOB, G., AND TANCA, L. 1990. *Logic Programming and Databases*. Springer, Berlin, Germany.

COURCELLE, B. 1987. An axiomatic definition of context-free rewriting and its application to NLC graph grammars. *Theor. Comput. Sci. 55*, 2-3, 141–181.

COURCELLE, B. 1990a. Graph rewriting: An algebraic and logic approach. In *Handbook of Theoretical Computer Science*, Vol. B. Elsevier Science Publishers, Amsterdam, The Netherlands, 193–242.

COURCELLE, B. 1990b. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Inform. Comput. 85*, 1, 12–75.

DONER, J. 1970. Tree acceptors and some of their applications. *J. Comput. Syst. Sci. 4*, 5, 406–451.

DOWLING, W. F. AND GALLIER, J. H. 1984. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *J. Log. Program. 1*, 3, 267–284.

DOWNEY, R. G. AND FELLOWS, M. R. 1999. *Parameterized Complexity*. Springer, New York, NY.

EBBINGHAUS, H.-D. AND FLUM, J. 1999. *Finite Model Theory*, 2nd ed. Springer Monographs in Mathematics. Springer, Berlin, Germany.

EITER, T. AND GOTTLOB, G. 1995. The complexity of logic-based abduction. *J. ACM 42*, 1, 3–42.

EITER, T., GOTTLOB, G., AND VEITH, H. 1997a. Generalized quantifiers in logic programs. In *Proceedings of the ESSLLI Workshop*. Lecture Notes in Computer Science, vol. 1754. Springer, Berlin, Germany, 72–98.

EITER, T., GOTTLOB, G., AND VEITH, H. 1997b. Modular logic programming and generalized quantifiers. In *Proceedings of LPNMR*. Lecture Notes in Computer Science, vol. 1265. Springer, Berlin, Germany, 290–309.

EITER, T., IANNI, G., SCHINDLAUER, R., AND TOMPITS, H. 2005. A uniform integration of higher-order reasoning and external evaluations in answer-set programming. In *Proceedings of IJCAI*. 90–96. www.jcai.org.

EITER, T., IANNI, G., SCHINDLAUER, R., AND TOMPITS, H. 2006. Effective integration of declarative rules with external evaluations for semantic-Web reasoning. In *Proceedings of ESWC*. Lecture Notes in Computer Science, vol. 4011. Springer, 273–287.

FILÉ, G. 1985. Tree automata and logic programs. In *Proceedings of STACS*. Lecture Notes in Computer Science, vol. 182. Springer, Berlin, Germany, 119–130.

FLUM, J., FRICK, M., AND GROHE, M. 2002. Query evaluation via tree-decompositions. *J. ACM 49*, 6, 716–752.

FLUM, J. AND GROHE, M. 2006. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. Springer, Berlin, Germany.

FOUSTOUCOS, E. AND GUESSARIAN, I. 2006. Complexity of monadic inf-datalog. application to temporal logic. CoRR abs/cs/0603122. http://arxiv.org/abs/cs/0603122.

FRICK, M. AND GROHE, M. 2004. The complexity of first-order and monadic second-order logic revisited. *Ann. Pure Appl. Logic 130*, 1-3, 3–31.

GOTTLOB, G. 1997. Relativized logspace and generalized quantifiers over finite ordered structures. *J. Symb. Log. 62*, 2, 545–574.

GOTTLOB, G., GRÄDEL, E., AND VEITH, H. 2002. Datalog LITE: A deductive query language with linear time model checking. *ACM Trans. Comput. Logic 3*, 1, 42–79.

GOTTLOB, G. AND KOCH, C. 2004. Monadic datalog and the expressive power of languages for Web information extraction. *J. ACM 51*, 1, 74–113.

GOTTLOB, G., PICHLER, R., AND WEI, F. 2006a. Bounded treewidth as a key to tractability of knowledge representation and reasoning. In *Proceedings of AAAI*. AAAI Press, Menlo Park, CA, 250–256.

GOTTLOB, G., PICHLER, R., AND WEI, F. 2006b. Tractable database design through bounded treewidth. In *Proceedings of PODS*. ACM Press, New York, NY, 124–133.

GOTTLOB, G., PICHLER, R., AND WEI, F. 2007. Monadic datalog over finite structures with bounded treewidth. In *Proceedings of PODS*. ACM Press, New York, NY, 165–174.

GOTTLOB, G., PICHLER, R., AND WEI, F. 2008. Abduction with bounded treewidth: From theoretical tractability to practically efficient computation. In *Proceedings of AAAI*. AAAI Press, Menlo Park, CA, 1541–1546.

GROHE, M. 1999. Descriptive and parameterized complexity. In *Proceedings of CSL*. Lecture Notes in Computer Science, vol. 1683. Springer, Berlin, Germany, 14–31.

GUSTEDT, J., MÆHLE, O. A., AND TELLE, J. A. 2002. The treewidth of Java programs. In *Proceedings of (ALENEX). 4th International Workshop on Algorithm Engineering and Experiments, Revised Papers*. Lecture Notes in Computer Science, vol. 2409. Springer, Berlin, Germany, 86–97.

HERRE, H., KRYNICKI, M., PINUS, A., AND VÄÄNÄNEN, J. A. 1991. The Härtig quantifier: A survey. *J. Symb. Log. 56*, 4, 1153–1183.

JAKL, M., PICHLER, R., AND WOLTRAN, S. 2009. Answer-set programming with bounded treewidth. In *Proceedings of IJCAI*. 816–822.

KLARLUND, N., MØLLER, A., AND SCHWARTZBACH, M. I. 2002. MONA implementation secrets. *Int. J. Found. Comput. Sci. 13*, 4, 571–586. Earlier version in *Proceedings of CIAA*, Lecture Notes in Computer Science, vol. 2088, Berlin, Germany.

KLOKS, T. 1994. *Treewidth: Computations and Approximations*. Springer, Berlin, Germany.

KOSTER, A. M. C. A., BODLAENDER, H. L., AND VAN HOESEL, S. P. M. 2001. Treewidth: Computational experiments. *Electron. Notes Discrete Math. 8*, 54–57.

LEONE, N., PFEIFER, G., FABER, W., EITER, T., GOTTLOB, G., PERRI, S., AND SCARCELLO, F. 2006. The DLV system for knowledge representation and reasoning. *ACM Trans. Comput. Log. 7*, 3, 499–562.

LIBKIN, L. 2004. *Elements of Finite Model Theory*. Texts in Theoretical Computer Science. Springer, Berlin, Germany.

MAKOWSKY, J. A. 2004. Algorithmic uses of the Feferman-Vaught Theorem. *Ann. Pure Appl. Logic 126*, 1-3, 159–213.

MANNILA, H. AND RÄIHÄ;, K.-J. 1992. *The design of relational databases*. Addison-Wesley, Reading, MA.

MARQUE-PUCHEU, G. 1983. Rational set of trees and the algebraic semantics of logic programming. *Acta Inform. 20*, 249–260.

MARYNS, H. 2006. On the implementation of tree automata: Limitations of the naive approach. In *Proceedings of the TLT. 5th International Treebanks and Linguistic Theories Conference (TLT)*. 235–246.

MINOUX, M. 1988. LTUR: A simplified linear-time unit resolution algorithm for Horn formulae and computer implementation. *Inform. Process. Lett. 29*, 1, 1–12.

NEVEN, F. AND SCHWENTICK, T. 2002. Query automata over finite trees. *Theor. Comput. Sci. 275*, 1-2, 633–674.

PALÙ, A. D., DOVIER, A., PONTELLI, E., AND ROSSI, G. 2009. Answer set programming with constraints using lazy grounding. In *Proceedings of ICLP*. 115–129.

SHER, G. 1997. Partially-ordered (branching) generalized quantifiers: A general definition. *J. Phil. Log. 26*, 1–43.

THATCHER, J. W. AND WRIGHT, J. B. 1968. Generalized finite automata theory with an application to a decision problem of second-order logic. *Math. Syst. Theor. 2*, 1, 57–81.

THOMAS, W. 1997. Languages, automata, and logic. In *Handbook of Formal Languages*, Vol. III. Springer, New York, NY, 389–455.

THORUP, M. 1998. All structured programs have small tree-width and good register allocation. *Inform. Comput. 142*, 2, 159–181.

ULLMAN, J. D. 1989. *Principles of Database and Knowledge-Base Systems*, Vol. 1. Computer Science Press, New York, NY.

VAN DEN EIJKHOF, F., BODLAENDER, H. L., AND KOSTER, A. M. C. A. 2007. Safe reduction rules for weighted treewidth. *Algorithmica 47*, 2, 139–158.

VARDI, M. Y. 1982. The complexity of relational query languages (extended abstract). In *Proceedings of STOC*. ACM Press, New York, NY, 137–146.

Received October 2008; revised September 2009; accepted September 2009