Monadic Datalog over Finite Structures with Bounded Treewidth

Georg Gottlob
Computing Laboratory
Information Systems Institute
Oxford University, Oxford OX1 3QD,UK
georg.gottlob@comlab.ox.ac.uk

Reinhard Pichler, and Fang Wei
Database and Artificial Intelligence Group
Technische Universitat Wien
A1040 Vienna, Austria
{pichler,wei}@dbai.tuwien.ac.at

ABSTRACT

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: 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 structure plus the tree decomposition. Moreover, we show that the resulting fragment of datalog can be evaluated in linear time (both w.r.t. the program size and w.r.t. the data size). This new approach is put to work by devising a new algorithm for the PRIMALITY problem (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 [**Nonnumerical Algorithms and Problems**]: Complexity of proof procedures, Computations on discrete structures; F.4.1 [**Mathematical Logic**]: Computational Logic

General Terms: Algorithms, Performance, Theory

Keywords: Tree decomposition, Treewidth, Fixed-parameter tractability, Monadic Second Order Logic, Datalog

1. INTRODUCTION

Over the past decade, parameterized complexity has evolved as an important subdiscipline in the field of computational complexity, see [8, 14]. 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 [5]: 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 [2, 13]. The idea is to first translate the MSO evaluation problem over finite structures into an equivalent MSO evaluation problem over colored binary trees. This problem can then be solved via the correspondence between MSO over trees and finite tree automata (FTA), see [25, 6]. 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 [22]. Consequently, it was already stated in [18] 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 propose monadic datalog (i.e., datalog where all intentional predicate symbols are unary) as a practical tool 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 structure plus the tree decomposition. 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 w.r.t. the program size and w.r.t. 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 with bounded treewidth generalizes a result from [15] where it was shown that MSO on trees has the same expressive

PODS'07, June 11–14, 2007, Beijing, China.

power as monadic datalog on trees. Several obstacles had to be overcome to prove this 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 hand.
- 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 [15] (notably several results on MSO-equivalences of tree structures shown in [24]) 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 [8, 13]) 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 monadic datalog.

In the second part of this paper, we put monadic datalog to work by presenting a new algorithm for the PRIMALITY problem (i.e., testing if some attribute in a relational schema is part of a key), which is well-known to be intractable (cf. [21]). In [17], PRIMALITY was shown to be MSO expressible. Hence, if the (incidence graph of the) relational schema has bounded treewidth, then this problem becomes tractable. However, two attempts to tackle this problem via the standard MSO-to-FTA approach turned out to be very problematical: We experimented with a prototype implementation using MONA (see [19]) for the MSO model checking. But we ended up with "memory leak" already for really small input data (see Section 6). Alternatively, we made an attempt to directly implement the MSO-to-FTA mapping proposed in [13]. However, the "state explosion" of the resulting FTA – which tends to occur already for comparatively simple formulae (cf. [22]) – led to failure yet 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. [4]). 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 w.r.t. the size of the MSO-formula and the treewidth. However, as our PRIMALITY program demonstrates, many relevant properties can be expressed by really short programs. 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 non-monadic program.
- Required transformations. The problem of a "state explosion" reported in [22] already refers to the transformation of (relatively simple) MSO formulae *on trees* to an FTA. If we consider MSO *on structures with 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 [13]) 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 if they admit an efficient implementation by the interpreter. Another example of a useful language extension is the introduction of generalized quantifiers. For the theoretical background of this concept, see [11, 12].

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 paper is organized as follows. After recalling some basic notions and results in Section 2, we prove 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 with bounded treewidth can be transformed into an equivalent monadic datalog program. In Section 5, we put monadic datalog to work by presenting a new FPT algorithm for the PRIMALITY problem in case that the relational schema has 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 Finite Structures and Treewidth

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 finite 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.

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.

The third condition 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 with treewidth 1.

For given $w \geq 1$, it can be decided in linear time if some structure has treewidth $\geq w$. Moreover, in case of a positive answer, a tree decomposition of width w can be computed in linear time, see [3].

In this paper, we consider the following form of *normalized tree decompositions*: (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*.

W.l.o.g., we restrict ourselves to the case that the domain $dom(\mathcal{A})$ has at least $w+1$ elements. Similarly to the normal form introduced in Theorem 6.72 of [8], it can be shown that any tree decomposition can be transformed in linear time into the above normalized form.

2.2 Monadic Second Order Logic

We assume some familiarity with Monadic Second Order logic (MSO), see e.g. [9, 20]. MSO extends First Order logic (FO) by the use of *set variables* (usually denoted by upper case letters), which range over sets of domain elements. In contrast, the *individual variables* (which are usually denoted by lower case letters) range over single domain elements. An MSO-formula φ over a τ-structure has as atomic formulae either atoms with some predicate symbol from τ or equality atoms. 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".

We call two structures $(\mathcal{A}, \bar{a})$ and $(\mathcal{B}, \bar{b})$ *k-equivalent* and write $(\mathcal{A}, \bar{a}) \equiv_k^{MSO} (\mathcal{B}, \bar{b})$, iff for every MSO-formula φ of quantifier depth $\leq k$, 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, which are played by two players – the spoiler and the duplicator: Two structures $(\mathcal{A}, \bar{a})$ and $(\mathcal{B}, \bar{b})$ are k-equivalent iff the duplicator has a winning strategy in the k-round MSO-game on $(\mathcal{A}, \bar{a})$ and $(\mathcal{B}, \bar{b})$, see e.g. [9, 20].

2.3 Datalog

We assume some familiarity with datalog, see e.g. [1, 4, 26]. Syntactically, a datalog program $\mathcal{P}$ is a set of function-free Horn clauses. The (minimal-model) semantics can be defined as the least fixpoint of applying the immediate consequence operator. 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 intentional.

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 $R_i(\bar{a}) \in \mathcal{E}(\mathcal{A})$ iff $\bar{a} \in R_i^{\mathcal{A}}$.

Evaluating a datalog program $\mathcal{P}$ over a structure $\mathcal{A}$ comes down to computing the least fixpoint of $\mathcal{P} \cup \mathcal{A}$. Concerning the complexity of datalog, we are mainly interested in the combined complexity (i.e., the complexity w.r.t. the size of the program $\mathcal{P}$ plus the size of the data $\mathcal{A}$). In general, the combined complexity of datalog is EXPTIME-complete (implicit in [27]). However, there are some fragments which can be evaluated much more efficiently. (1) *Propositional datalog* (i.e., all rules are ground) can be evaluated in linear time (combined complexity), see [7, 23]. (2) The *guarded fragment* of datalog (i.e., every rule r contains an extensional atom B in the body, s.t. all variables occurring in r also occur in B) can be evaluated in time $\mathcal{O}(|\mathcal{P}| * |\mathcal{A}|)$. (3) *Monadic datalog* (i.e., all intentional predicates are unary) is NP-complete (combined complexity), see [15].

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 3.3 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 [24], 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 finite structure. Then we define $\mathcal{T}_t = \langle T_t, (A_s)_{s \in T_t} \rangle$ and $\bar{\mathcal{T}}_t = \langle \bar{T}_t, (A_s)_{s \in \bar{T}_t} \rangle$.

DEFINITION 3.2. *Let $\mathcal{A}$ be a finite 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 $\mathcal{S}$.

DEFINITION 3.3. *Let $w \geq 1$ be a natural number and let $\mathcal{A}$ and $\mathcal{B}$ be finite structures over some signature τ. Moreover, let $(a_0, \ldots, a_w)$ (resp. $(b_0, \ldots, b_w)$) be a tuple of pairwise distinct elements in A (resp. B).*

We call $(a_0, \ldots, a_w)$ and $(b_0, \ldots, b_w)$ equivalent *and write $(a_0, \ldots, a_w) \equiv (b_0, \ldots, b_w)$, iff 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.*

We are now ready to generalize results from [24] (dealing with trees plus a distinguished node) to the case of finite 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 3.4. *Let $\mathcal{A}$ and $\mathcal{B}$ be τ-structures, let $\mathcal{S}$ (resp. $\mathcal{T}$) be a normalized tree decomposition of $\mathcal{A}$ (resp. of $\mathcal{B}$) of width w, and let s (resp. t) be an internal node in $\mathcal{S}$ (resp. in $\mathcal{T}$).*

(1) permutation nodes. *Let s' (resp. t') be the only child of s in $\mathcal{S}$ (resp. 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, s.t. $\bar{a} = \pi(\bar{a}')$ 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' (resp. t') be the only child of s in $\mathcal{S}$ (resp. 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 (resp. t_1 and t_2) be the children of s in $\mathcal{S}$ (resp. 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 3.5. *Let $\mathcal{A}$ and $\mathcal{B}$ be τ-structures, let $\mathcal{S}$ (resp. $\mathcal{T}$) be a normalized tree decomposition of $\mathcal{A}$ (resp. of $\mathcal{B}$) of width w, and let s (resp. t) be an internal node in $\mathcal{S}$ (resp. in $\mathcal{T}$).*

(1) permutation nodes. *Let s' (resp. t') be the only child of s in $\mathcal{S}$ (resp. 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, s.t. $\bar{a} = \pi(\bar{a}')$ 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' (resp. t') be the only child of s in $\mathcal{S}$ (resp. 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 (resp. t_1 and t_2) be the children of s in $\mathcal{S}$ (resp. 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 3.6. *Let $\mathcal{A}$ and $\mathcal{B}$ be τ-structures, let $\mathcal{S}$ (resp. $\mathcal{T}$) be a normalized tree decomposition of $\mathcal{A}$ (resp. of $\mathcal{B}$) of width w, and let s (resp. t) be an arbitrary node in $\mathcal{S}$ (resp. in $\mathcal{T}$), whose bag is $(a_0, \ldots, a_w)$ (resp. $(b_0, \ldots, b_w)$).*

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 $i \in \{0, \ldots, w\}$.

DISCUSSION. Lemma 3.4 provides the intuition how to determine the k-type of the substructure induced by a subtree $\mathcal{S}_s$ via a bottom-up traversal of the tree decomposition $\mathcal{S}$. The three cases in the lemma refer to the three kinds of nodes which the root node s of this subtree can have. The essence of the lemma is that the type of the structure induced by $\mathcal{S}_s$ is fully determined by the type of the structure induced by the subtree rooted at the child node(s) plus the relations between elements in the bag at node s. Of course, this is no big surprise. Analogously, Lemma 3.5 deals with the k-type of the substructure induced by a subtree $\bar{\mathcal{S}}_s$, which can be obtained via a top-down traversal of $\mathcal{S}$. Finally, Lemma 3.6 shows how 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.

PROOF IDEA OF THE LEMMAS. The proof of the three lemmas is by Ehrenfeucht-Fraïssé games (see [9, 20]). 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.

4. MONADIC DATALOG

In this section, we introduce two restricted fragments of datalog, namely *monadic datalog* over finite structures with bounded treewidth and the *quasi-guarded fragment* of Datalog. Let $\tau = \{R_1, \ldots, R_K\}$ be a set of predicate symbols and let $w \geq 1$ denote the treewidth. Then 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 with treewidth w is a set of datalog rules where all extensional predicates are from τ_{td} and all intentional predicates are unary.*

For any τ-structure $\mathcal{A}$ with normalized tree decomposition $\mathcal{T} = \langle T, (A_t)_{t \in T} \rangle$ of width w, we denote by $\mathcal{A}_{td}$ the τ_{td}-structure representing $\mathcal{A}$ plus $\mathcal{T}$ as follows: The domain of $\mathcal{A}_{td}$ is the union of $dom(\mathcal{A})$ and the 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 $\mathcal{T}$. By [3], one can compute $\mathcal{A}_{td}$ from $\mathcal{A}$ in linear time w.r.t. the size of $\mathcal{A}$. Hence, the size (of some reasonable encoding, see e.g. [13]) of $\mathcal{A}_{td}$ is also linearly bounded by the size of $\mathcal{A}$.

As we recalled in Section 2.3, 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 "quasi-guarded". The evaluation of this fragment can be easily shown to be tractable.

DEFINITION 4.2. *Let B be an atom and y a variable in some rule r. We call y "functionally dependent" on B if in every ground instantiation r' of r, the value of y is uniquely determined by the value of B.*

We call a datalog program $\mathcal{P}$ "quasi-guarded" if every rule r contains an extensional atom B, s.t. every variable occurring in r either occurs in B or is functionally dependent on B.

THEOREM 4.3. *Let $\mathcal{P}$ be a quasi-guarded datalog program and let $\mathcal{A}$ be a finite 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 "quasi-guard" of r, i.e., all variables in r either occur in B or are functionally dependent on B. In order to compute all possible ground instances r' of r over $\mathcal{A}$, we first instantiate B. The maximal number of such instantiations is clearly bounded by $|\mathcal{A}|$. Since all other variables occurring in r are functionally dependent on the variables in B, in fact the number of all possible ground instantiations r' of r is bounded by $|\mathcal{A}|$.

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 2.3, 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 with 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 $\{a_0, \ldots, a_w\}$, i.e.,

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

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

THEOREM 4.4. *Let τ and $w \geq 1$ be arbitrary but fixed. Every MSO-definable unary query over τ-structures of treewidth w is also definable in the quasi-guarded fragment of monadic datalog 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.

W.l.o.g., we only consider the case of structures whose domain has $\geq w + 1$ elements. We maintain two disjoint sets of k-types $\Theta^{\uparrow}$ 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 is the root of $\mathcal{S}$ while, for $\Theta^{\downarrow}$, we require that s is a leaf node of $\mathcal{T}$. We maintain for each type ϑ a witness $W(\vartheta) = \langle \mathcal{A}, \mathcal{T}, s \rangle$. The types in $\Theta^{\uparrow}$ and $\Theta^{\downarrow}$ will serve as predicate names in the monadic datalog program to be constructed. Initially, $\Theta^{\uparrow} = \Theta^{\downarrow} = \mathcal{P} = \emptyset$.

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

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

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

INDUCTION STEP. We construct new structures by extending the tree decompositions of existing witnesses in "bottom-up" direction, i.e., by introducing a new root node. This root node may be one of three kinds of nodes.

(a) Permutation nodes. For each $\vartheta' \in \Theta^{\uparrow}$, let $W(\vartheta') = \langle \mathcal{A}, \mathcal{S}', s' \rangle$ 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, s.t. s is a permutation node, i.e., there exists some permutation π, s.t. $A_s = (a_{\pi(0)}, \ldots, a_{\pi(w)})$

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

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

(b) Element replacement nodes. For each $\vartheta' \in \Theta^{\uparrow}$, let $W(\vartheta') = \langle \mathcal{A}', \mathcal{S}', s' \rangle$ 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, s.t. 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 $dom(\mathcal{A}) = dom(A') \cup \{a_0\}$ where the EDB $\mathcal{E}(\mathcal{A}')$ is extended to the EDB $\mathcal{E}(\mathcal{A})$ by new ground atoms from $\mathcal{R}(\bar{a})$, s.t. 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 type $\vartheta \in \Theta^{\uparrow}$ with $W(\vartheta) = \langle \mathcal{B}, \mathcal{T}, t \rangle$, s.t. $(\mathcal{A}, s) \equiv_k^{MSO} (\mathcal{B}, t)$. If such a ϑ exists, we take it. Otherwise we invent a new token ϑ, add it to $\Theta^{\uparrow}$ and set $W(\vartheta) := \langle \mathcal{A}, \mathcal{S}, s \rangle$. In any case, we add the following rule to the program $\mathcal{P}$:

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

(c) Branch nodes. Let ϑ_1, ϑ_2 be two (not necessarily distinct) types in $\Theta^\uparrow$ with $W(\vartheta_1) = \langle \mathcal{A}_1, \mathcal{S}_1, s_1\rangle$ and $W(\vartheta_2) = \langle \mathcal{A}_2, \mathcal{S}_2, s_2\rangle$. Let $A_{s_1} = (a_0, \ldots, a_w)$ and $A_{s_2} = (b_0, \ldots, b_w)$, respectively. Moreover, let $dom(\mathcal{A}_1) \cap dom(\mathcal{A}_2) = \emptyset$.

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, i.e., $\mathcal{E}(\mathcal{A}_1) \cap \mathcal{R}(\bar{a}) \neq \mathcal{E}(\mathcal{A}'_2) \cap \mathcal{R}(\bar{a})$. 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 $A_s = A_{s_1} = A_{s'_2}$. By construction, $\mathcal{S}$ is a normalized tree decomposition of the structure $\mathcal{A}$ with $dom(\mathcal{A}) = dom(\mathcal{A}_1) \cup dom(\mathcal{A}'_2)$ 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 type $\vartheta \in \Theta^\uparrow$ with $W(\vartheta) = \langle \mathcal{B}, \mathcal{T}, t\rangle$, s.t. $(\mathcal{A}, s) \equiv_k^{MSO} (\mathcal{B}, t)$. If such a ϑ exists, we take it. Otherwise we invent a new token ϑ, add it to $\Theta^\uparrow$ and set $W(\vartheta) := \langle \mathcal{A}, \mathcal{S}, s\rangle$. In any case, we add the following rule to the program $\mathcal{P}$:

$$\begin{aligned}\vartheta(v) \leftarrow\ & bag(v, x_0, x_1, \ldots, x_w),\\ & child_1(v_1, v), \vartheta_1(v_1), child_2(v_2, v), \vartheta_2(v_2),\\ & bag(v_1, x_0, x_1, \ldots, x_w), bag(v_2, x_0, x_1, \ldots, x_w).\end{aligned}$$

2. *"Top-down" construction of $\Theta^\downarrow$.*

Analogously to the "bottom-up" construction of $\Theta^\uparrow$, we construct the set $\Theta^\downarrow$ of types 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 $W(\vartheta') = \langle \mathcal{A}', \mathcal{S}', s'\rangle$ of some already computed type $\vartheta' \in \Theta^\downarrow$. 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 $(\mathcal{A}, s)$.

3. *Element selection.*

We consider all pairs of types $\vartheta_1 \in \Theta^\uparrow$ and $\vartheta_2 \in \Theta^\downarrow$. Let $W(\vartheta_1) = \langle \mathcal{A}_1, \mathcal{S}_1, s_1\rangle$ and $W(\vartheta_2) = \langle \mathcal{A}_2, \mathcal{S}_2, s_2\rangle$. Moreover, let $A_{s_1} = (a_0, \ldots, a_w)$ and $A_{s_2} = (b_0, \ldots, b_w)$, respectively, and let $dom(\mathcal{A}_1) \cap dom(\mathcal{A}_2) = \emptyset$.

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, i.e., $\mathcal{E}(\mathcal{A}_1) \cap \mathcal{R}(\bar{a}) \neq \mathcal{E}(\mathcal{A}'_2) \cap \mathcal{R}(\bar{a})$. 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 $\mathcal{P}$.

$$\varphi(x_i) \leftarrow \vartheta_1(v), \vartheta_2(v), bag(v, x_0, \ldots, x_w).$$

We claim that the program $\mathcal{P}$ with distinguished monadic predicate φ is the desired monadic datalog program, i.e., 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: $\mathcal{A} \models \varphi(a)$ iff $\varphi(a)$ is in the fixpoint of $\mathcal{P} \cup \mathcal{A}_{td}$.

Note that the intentional predicates in $\Theta^\uparrow$, $\Theta^\downarrow$, and $\{\varphi\}$ are layered in that we can first compute the fixpoint of the predicates in $\Theta^\uparrow$, 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 3.4 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 $\vartheta \in \Theta^\uparrow$ if and only if the atom $\vartheta(s)$ is in the fixpoint of $\mathcal{P} \cup \mathcal{A}_{td}$. Again this can be shown by an easy induction argument using Lemma 3.4.

Analogously, we may conclude via Lemma 3.5 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 $\vartheta \in \Theta^\downarrow$ if and only if the atom $\vartheta(s)$ is in the fixpoint of $\mathcal{P} \cup \mathcal{A}_{td}$. The definition of the predicate φ in part 3 is a direct realization of Lemma 3.6. It thus follows that $\mathcal{A} \models \varphi(a)$ iff $\varphi(a)$ is in the fixpoint 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 quasi-guarded, i.e., they all contain an atom B with an extensional predicate, s.t. all other variables in this rule are functionally dependent on the variables in B. For instance, in the rule added to $\Theta^\uparrow$ in case of a branch node, the atom $bag(v, x_0, \ldots, x_w)$ is the quasi-guard. 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 4.4 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 [13]).

COROLLARY 4.5. *There exists an algorithm that solves the evaluation problem of unary MSO-queries $\varphi(x)$ over τ-structures $\mathcal{A}$ with treewidth w 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 4.4, we can construct an equivalent, quasi-guarded 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 [3], 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 4.3, the quasi-guarded 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. □

5. MONADIC DATALOG AT WORK

In this section, we put monadic datalog to work by constructing a new algorithm for the PRIMALITY problem (i.e., testing if some attribute in a relational schema is part of a key), which is well-known to be intractable (cf. [21]). By [17], this problem is expressible in MSO over appropriate structures and thus fixed-parameter tractable w.r.t. the treewidth. Below, we show that this problem admits a succinct and efficient solution via datalog.

5.1 The Primality Problem

Recall that 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. W.l.o.g., we only consider FDs whose right-hand side consists of a single attribute. Let $f \in F$ with $f : Y \to A$. We refer to $Y \subseteq R$ and $A \in R$ as $lhs(f)$ and $rhs(f)$, respectively. For any $X \subseteq R$, we write X^+ to denote the closure of X, i.e., the set of all attributes determined by X. An attribute A is contained in X^+ iff either $A \in X$ or there exists a "derivation sequence" of A from X in F of the form $X \to X \cup \{A_1\} \to X \cup \{A_1, A_2\} \to \ldots \to X \cup \{A_1, \ldots, A_n\}$, s.t. $A_n = A$ 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. [21]).

In this paper, 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. $lh(b, f)$ (resp. $rh(b, f)$) means that b occurs in $lhs(f)$ (resp. in $rhs(f)$). The treewidth of (R, F) is then defined as the treewidth of this τ-structure.

A relational schema (R, F) defines a hypergraph $H(R, F)$ whose vertices are the attributes 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 iff (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.

Before we present our datalog program solving the primality problem, we slightly modify the notion of normalized tree decompositions from Section 2.1. Recall that an *element replacement node* replaces exactly one element in the bag of the child node by a new element. For our algorithm, it is preferable to split this action into two steps, namely, an *element removal node* (which removes one element from the bag of its child node) and an *element introduction node* (which introduces one new element). With our representation of relational schemas (R, F) as finite structures, the domain elements are the attributes and FDs in (R, F). Hence, in total, the former element replacement nodes give rise to four kinds of nodes, namely, attribute removal nodes, FD removal nodes, attribute introduction nodes, and FD introduction nodes. Moreover, it is now preferable to consider the bags as a pair of sets At (of attributes) and Fd (of FDs) rather than as tuples. Hence, we may delete permutation nodes from the tree decomposition. 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 $rhs(f)$ is 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 1, we describe a datalog program, where the input is given as an attribute $a \in R$ and a τ_{td}-structure, s.t. a occurs in the bag at the root of the tree decomposition.

Program PRIMALITY

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

/* **internal node.** */

/* attribute introduction node. */
$solve(v, Y \uplus \{b\}, FY, C^o, \Delta C, FC) \leftarrow bag(v, At \uplus \{b\}, Fd),$
$\quad child_1(v_1, v),\ bag(v_1, At, Fd),\ solve(v_1, Y, FY, C^o, \Delta C, FC).$

$solve(v, Y, FY, C^o \uplus \{b\}, \Delta C, FC) \leftarrow bag(v, At \uplus \{b\}, Fd),$
$\quad child_1(v_1, v),\ bag(v_1, At, Fd),\ consistent(FC, C^o \uplus \{b\}),$
$\quad solve(v_1, Y, FY_1, C^o, \Delta C, FC),$
$\quad outside(FY_2, Y, At, Fd),\ FY = FY_1 \cup FY_2.$

/* FD introduction node. */
$solve(v, Y, FY, C^o, \Delta C, FC) \leftarrow bag(v, At, Fd \uplus \{f\}),$
$\quad child_1(v_1, v),\ bag(v_1, At, Fd),\ rh(b, f),\ b \in Y,$
$\quad solve(v_1, Y, FY, C^o, \Delta C, FC).$

$solve(v, Y, FY, C^o, \Delta C, FC) \leftarrow bag(v, At, Fd \uplus \{f\}),$
$\quad child_1(v_1, v),\ bag(v_1, At, Fd),\ rh(b, f),\ b \in C^o,$
$\quad solve(v_1, Y, FY_1, C^o, \Delta C, FC),$
$\quad outside(FY_2, Y, At, \{f\}),\ FY = FY_1 \cup FY_2.$

$solve(v, Y, FY, C^o, \Delta C \uplus \{b\}, FC \uplus \{f\}) \leftarrow bag(v, At, Fd \uplus \{f\}),$
$\quad child_1(v_1, v),\ bag(v_1, At, Fd),\ rh(b, f),\ b \in C^o,$
$\quad solve(v_1, Y, FY_1, C^o, \Delta C, FC),\ consistent(\{f\}, C^o),$
$\quad outside(FY_2, Y, At, \{f\}),\ FY = FY_1 \cup FY_2.$

/* attribute removal node. */
$solve(v, Y, FY, C^o, \Delta C, FC) \leftarrow bag(v, At, Fd),$
$\quad child_1(v_1, v),\ bag(v_1, At \uplus \{b\}, Fd),$
$\quad solve(v_1, Y \uplus \{b\}, FY, C^o, \Delta C, FC).$

$solve(v, Y, FY, C^o, \Delta C, FC) \leftarrow bag(v, At, Fd),$
$\quad child_1(v_1, v),\ bag(v_1, At \uplus \{b\}, Fd),$
$\quad solve(v_1, Y, FY, C^o \uplus \{b\}, \Delta C \uplus \{b\}, FC).$

/* FD removal node. */
$solve(v, Y, FY, C^o, \Delta C, FC) \leftarrow bag(v, At, Fd),$
$\quad child_1(v_1, v),\ bag(v_1, At, Fd \uplus \{f\}),\ rh(b, f),\ b \in Y,$
$\quad solve(v_1, Y, FY, C^o, \Delta C, FC).$

$solve(v, Y, FY, C^o, \Delta C, FC) \leftarrow bag(v, At, Fd),$
$\quad child_1(v_1, v),\ bag(v_1, At, Fd \uplus \{f\}),\ rh(b, f),\ b \in C^o,$
$\quad solve(v_1, Y, FY \uplus \{f\}, C^o, \Delta C, FC \uplus \{f\}).$

$solve(v, Y, FY, C^o, \Delta C, FC) \leftarrow bag(v, At, Fd),$
$\quad child_1(v_1, v),\ bag(v_1, At, Fd \uplus \{f\}),\ rh(b, f),\ b \in C^o,$
$\quad solve(v_1, Y, FY \uplus \{f\}, C^o, \Delta C, FC),\ f \notin FC.$

/* branch node. */
$solve(v, Y, FY_1 \cup FY_2, C^o, \Delta C_1 \cup \Delta C_2, FC) \leftarrow bag(v, At, Fd),$
$\quad child_1(v_1, v),\ bag(v_1, At, Fd),\ child_2(v_2, v),\ bag(v_2, At, Fd),$
$\quad solve(v_1, Y, FY_1, C^o, \Delta C_1, FC),$
$\quad solve(v_2, Y, FY_2, C^o, \Delta C_2, FC).$

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

Figure 1: Primality Test.

Some words on the notation used in this program are in order: We are using lower case letters v, 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. In contrast, upper case 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). Note that these (ordered) 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. Indeed, we have implemented these "fixed-size" sets by means of k-tuples with $k \leq (w+1)$ over $\{0,1\}$. For the sake of readability, we are using non-datalog expressions involving set operators $\uplus$ (disjoint union), $\cup$, $\cap$, $\subseteq$, $\in$. For the fixed-size (ordered) sets under consideration here, one could, of course, easily replace these operators by "proper" datalog expressions.

In order to facilitate the discussion of the PRIMALITY-program, we introduce the following notation. Let (R,F) be the input schema with tree decomposition $\mathcal{T}$. For any node v in $\mathcal{T}$, we write as usual $\mathcal{T}_v$ to denote the subtree of $\mathcal{T}$ rooted at v. By $FD(v)$ we denote the FDs in the bag of v while $FD(\mathcal{T}_v)$ denotes the FDs that occur in any bag in $\mathcal{T}_v$. Analogously, we write $Att(v)$ and $Att(\mathcal{T}_v)$ as a short-hand for the attributes occurring in the bag of v respectively in any bag in $\mathcal{T}_v$.

Our PRIMALITY-program checks the primality of a by searching for an attribute set $\mathcal{Y} \subseteq R$, s.t. $\mathcal{Y}$ is closed w.r.t. 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. This is clearly a sufficient and necessary condition for a to be prime.

At the heart of our PRIMALITY-program is the intentional predicate $solve(v, Y, FY, C^o, \Delta C, FC)$ with the following intended meaning: v denotes a node in $\mathcal{T}$. Y (resp. C^o) is the projection of $\mathcal{Y}$ (resp. of $R \setminus \mathcal{Y}$) onto $Att(v)$. We consider $R \setminus \mathcal{Y}$ as ordered w.r.t. an appropriate derivation sequence of R from $\mathcal{Y} \cup \{a\}$, i.e., suppose that $\mathcal{Y} \cup \{A_0\} \to \mathcal{Y} \cup \{A_0, A_1\} \to \mathcal{Y} \cup \{A_0, A_1, A_2\} \to \ldots \to \mathcal{Y} \cup \{A_0, A_1, \ldots, A_n\}$, s.t. $A_0 = a$ and $\mathcal{Y} \cup \{A_0, A_1, \ldots, A_n\} = R$. W.l.o.g., the A_i's may be assumed to be pairwise distinct. Then for any two $i \neq j$, we simply set $A_i < A_j$ iff $i < j$. 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 intended meaning of the set FC is that it contains those FDs in $FD(v)$ which are used in the above derivation sequence. Informally, FY contains those FDs in $FD(v)$ for which we have already verified that they do not constitute a contradiction with the closedness of $\mathcal{Y}$. Finally, ΔC contains those attributes from $Att(v)$ for which we have already shown that they can be derived from $\mathcal{Y}$ plus smaller atoms in C^o. More precisely, for all values $v, Y, FY, C^o, \Delta C, FC$, the ground fact $solve(v, Y, FY, C^o, \Delta C, FC)$ shall be in the fixpoint of the program, iff the following condition holds:

PROPERTY A. There exist extensions $\overline{Y}$ of Y and $\overline{C}^o$ of C^o to $Att(\mathcal{T}_v)$ and an extension $\overline{FC}$ of FC to $FD(\mathcal{T}_v)$, s.t.

1. $\overline{Y}$ and $\overline{C}^o$ form a partition of $Att(\mathcal{T}_v)$,
2. $\forall f \in FD(\mathcal{T}_v) \setminus FD(v)$, if $rhs(f) \notin \overline{Y}$, then $lhs(f) \not\subseteq \overline{Y}$. Moreover, $FY = \{f \in FD(v) \mid rhs(f) \notin \overline{Y}$ and $lhs(f) \cap Att(\mathcal{T}_v) \not\subseteq \overline{Y}\}$.
3. $\forall f \in \overline{FC}$, f is consistent with the order on $\overline{C}^o$, i.e., $\forall f \in \overline{FC}$, $\forall b \in lhs(f) \cap \overline{C}^o$: $b < rhs(f)$ holds.
4. $\Delta C \cup \overline{C}^o \setminus Att(v) = \{rhs(f) \mid f \in \overline{FC}\}$,

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

- $outside(FY, Y, At, Fd)$ is in the fixpoint iff $FY = \{f \in Fd \mid rhs(f) \notin Y$ and $lhs(f) \cap At \not\subseteq Y\}$, i.e., 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 in the fixpoint iff $\forall f \in FC$ we have $rhs(f) \in C^o$ and $\forall b \in lhs(f) \cap C^o$: $b < rhs(f)$, i.e., the FDs in FC are only used to derive greater attributes from smaller ones (plus attributes from $\mathcal{Y}$).
- 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.

THEOREM 5.1. *The datalog program in Figure 1 decides the primality problem for a fixed attribute a, i.e., the fact "success" is in the fixpoint of this program iff the input τ_{td}-structure encodes a relational schema (R,F), s.t. a is part of a key.*

Moreover, for any relational schema (R,F) with treewidth w, the computation of the τ_{td}-structure and the evaluation of the datalog program can be done in time $\mathcal{O}(f(w) * |(R,F)|)$ for some function f.

PROOF. Suppose that the predicate *solve* indeed has the meaning described above. Then the rule with head *success* reads as follows: *success* is in the fixpoint, iff v denotes the root of $\mathcal{T}$, a is an attribute in the bag at v, and Y is the projection of the desired attribute set $\mathcal{Y}$ onto $Att(v)$, i.e., (1) $\mathcal{Y}$ is closed (this is ensured by the condition that $\{f \in Fd \mid rhs(f) \notin Y\} = FY$), $a \notin \mathcal{Y}$ and, finally, all attributes in $R \setminus (\mathcal{Y} \cup \{a\})$ are indeed determined by $\mathcal{Y} \cup \{a\}$ (this is ensured by the condition $\Delta C = C^o \setminus \{a\}$). Moreover, it is straightforward to prove the correctness of the *solve* predicate by structural induction on $\mathcal{T}$.

For the linear time data complexity, the crucial observation is that our program in Figure 1 is essentially a succinct representation of a quasi-guarded monadic datalog program. For instance, in the atom $solve(v, Y, FY, C^o, \Delta C, FC)$, the (ordered) sets Y, FY, C^o, ΔC, and FC are subsets of the bag of v. Hence, each combination Y, FY, C^o, ΔC, FC could be represented by 5 subsets resp. tuples $r_1, \ldots, r_5$ over $\{0, \ldots, w\}$ referring to indices of elements in the bag of v. Recall that w is a fixed constant. Hence, $solve(v, Y, FY, C^o, \Delta C, FC)$, is simply a succinct representation of constantly many monadic predicates of the form $solve_{\langle r_1,\ldots,r_5\rangle}(v)$. The quasi-guard in each rule is $bag(v, At, Fd)$ (possibly extended by a disjoint union with $\{b\}$ or $\{f\}$, respectively). Thus, the linear time bound follows immediately from Theorem 4.3. □

5.2 Primality as Monadic Predicate

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, s.t. 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 1. Obviously, this method has *quadratic* time complexity w.r.t. the data size. However, in this section, we describe a *linear* time algorithm. For this purpose, we further modify the notion of normalized tree decompositions from Section 5.1:

(1) We need an additional extensional predicate *dist* with the intended meaning that for every $a \in R$, we uniquely

choose one distinguished node v in $\mathcal{T}$ with $a \in Att(v)$. This choice is denoted by $dist(a, v)$. W.l.o.g., we assume that v is a leaf node since, otherwise, we simply select a node v containing a in its bag, attach a copy of v as new child of v and re-apply the normalization from Section 2.1 and 5.1. The *dist* predicate can be computed as part of the construction of the tree decomposition within the linear time bound of the computation of $\mathcal{T}$.

(2) For every branch node v in the tree decomposition, we insert a new node u as new parent of v, s.t. u and v have identical bags. Hence, together with the two child nodes of v, 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 A in Section 5.1, we also compute a predicate *solve*$\downarrow$, whose meaning is described by replacing every occurrence of $\mathcal{T}_v$ in Property A by $\bar{\mathcal{T}}_v$. 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*$\downarrow(v, \dots)$ for a leaf node v of $\mathcal{T}$ is exactly the same as if we computed $solve(v, \dots)$ for the tree rooted at v. Hence, we can define the predicate *prime*() as follows.

Program Monadic-Primality

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

By the intended meaning of *solve*$\downarrow$ and by the properties of the Primality algorithm in Section 5.1, we immediately get the following result.

THEOREM 5.2. *The monadic predicate prime() as defined above selects precisely the prime attributes. Moreover, it can be computed in linear time w.r.t. the size of the input structure.*

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 5.1 in C++. The experiments were conducted on Linux kernel 2.6.17 with an 1.60GHz 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.

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.

EXPERIMENTAL RESULTS. The outcome of the tests is shown in Table 1, where tw stands for the treewidth; #Att, #FD, and #tn stand for the number of attributes, FDs, and tree nodes, respectively. The processing time (in ms) obtained with our C++ implementation following the monadic datalog program in Section 5.1 are displayed in the column labelled "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.

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	–

Table 1: Processing Time in ms for PRIMALITY.

In [16], we proved the FPT of several non-monotonic reasoning problems via Courcelle's Theorem. Moreover, we also carried out some experiments with a prototype implementation using MONA (see [19]) 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 1. Due to problems discussed in [16], MONA does not ensure linear data complexity. Hence, all testes below line 3 of the table failed with "memory leak". Moreover, also in cases where the exponential data complexity does not yet "hurt", our datalog approach outperforms the MSO-to-FTA approach by a factor of 1000 or even more.

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

(1) Succinct representation by non-monadic datalog. As was mentioned in the proof sketch of Theorem 5.1, our datalog program can be regarded as a succinct representation of a big monadic datalog program. If all possible ground instances of our datalog rules had to be materialized, then we would end up with a ground program of the same size as with the equivalent monadic program. However, it turns out that the vast majority of possible instantiations is never computed since they are not "reachable" along the bottom-up computation.

(2) General optimizations and lazy grounding. In principle, our implementation follows 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 4.3. A further improvement is achieved by the natural idea of generating only those ground instances of rules which actually produce new facts.

(3) Language extensions. As was mentioned in Section 5.1, we are using language constructs (in particular, for handling sets of attributes and FDs) which are not part of the datalog language. 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.

(4) Further improvements. We are planning to implement further improvements. For instance, we are currently apply-

ing a strict bottom-up intuition as we compute new facts $solve(v, \dots)$. 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 with 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 PRIMALITY problem of relational schemas with 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 [10]) is much more involved. A monadic datalog program solving the relevance problem also in this general case will be presented in a forthcoming paper.

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). Free access to this tool will be provided on the Web.

8. REFERENCES

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

[2] S. Arnborg, J. Lagergren, and D. Seese. Easy Problems for Tree-Decomposable Graphs. *J. Algorithms*, 12(2):308–340, 1991.

[3] H. L. Bodlaender. A Linear-Time Algorithm for Finding Tree-Decompositions of Small Treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.

[4] S. Ceri, G. Gottlob, and L. Tanca. *Logic Programming and Databases*. Springer, 1990.

[5] B. Courcelle. Graph Rewriting: An Algebraic and Logic Approach. In *Handbook of Theoretical Computer Science, Volume B*, pages 193–242. Elsevier Science Publishers, 1990.

[6] J. Doner. Tree acceptors and some of their applications. *J. Comput. Syst. Sci.*, 4(5):406–451, 1970.

[7] W. F. Dowling and J. H. Gallier. Linear-Time Algorithms for Testing the Satisfiability of Propositional Horn Formulae. *J. Log. Program.*, 1(3):267–284, 1984.

[8] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, New York, 1999.

[9] H.-D. Ebbinghaus and J. Flum. *Finite Model Theory, 2nd edition*. Springer Monographs in Mathematics. Springer, 1999.

[10] T. Eiter and G. Gottlob. The Complexity of Logic-Based Abduction. *J. ACM*, 42(1):3–42, 1995.

[11] T. Eiter, G. Gottlob, and H. Veith. Generalized quantifiers in logic programs. In *ESSLLI'97 Workshop*, volume 1754 of *LNCS*, pages 72–98. Springer, 1997.

[12] T. Eiter, G. Gottlob, and H. Veith. Modular logic programming and generalized quantifiers. In *Proc. LPNMR'97*, volume 1265 of *LNCS*, pages 290–309, 1997.

[13] J. Flum, M. Frick, and M. Grohe. Query evaluation via tree-decompositions. *J. ACM*, 49(6):716–752, 2002.

[14] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. Springer, 2006.

[15] G. Gottlob and C. Koch. Monadic datalog and the expressive power of languages for Web information extraction. *J. ACM*, 51(1):74–113, 2004.

[16] G. Gottlob, R. Pichler, and F. Wei. Bounded Treewidth as a Key to Tractability of Knowledge Representation and Reasoning. In *Proc. AAAI 2006*, pages 250–256. AAAI Press, 2006.

[17] G. Gottlob, R. Pichler, and F. Wei. Tractable database design through bounded treewidth. In *Proceedings of the Twenty-Fifth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 2006)*, pages 124–133. ACM, 2006.

[18] M. Grohe. Descriptive and Parameterized Complexity. In *Proc. CSL'99*, volume 1683 of *LNCS*, pages 14–31. Springer, 1999.

[19] N. Klarlund, A. Møller, and M. I. Schwartzbach. MONA Implementation Secrets. *International Journal of Foundations of Computer Science*, 13(4):571–586, 2002. World Scientific Publishing Company. Earlier version in Proc. CIAA'00, LNCS vol. 2088.

[20] L. Libkin. *Elements of Finite Model Theory*. Texts in Theoretical Computer Science. Springer, 2004.

[21] H. Mannila and K.-J. Räihä;. *The design of relational databases*. Addison-Wesley, 1992.

[22] H. Maryns. On the Implementation of Tree Automata: Limitations of the Naive Approach. In *Proc. 5th Int. Treebanks and Linguistic Theories Conference (TLT 2006)*, pages 235–246, 2006.

[23] M. Minoux. LTUR: A Simplified Linear-Time Unit Resolution Algorithm for Horn Formulae and Computer Implementation. *Inf. Process. Lett.*, 29(1):1–12, 1988.

[24] F. Neven and T. Schwentick. Query automata over finite trees. *Theoretical Computer Science*, 275(1-2):633–674, 2002.

[25] J. W. Thatcher and J. B. Wright. Generalized Finite Automata Theory with an Application to a Decision Problem of Second-Order Logic. *Mathematical Systems Theory*, 2(1):57–81, 1968.

[26] J. D. Ullman. *Principles of Database and Knowledge-Base Systems, Vol. 1*. Computer Science Press, 1989.

[27] M. Y. Vardi. The complexity of relational query languages (extended abstract). In *Proc. STOC'82*, pages 137–146. ACM, 1982.