Containment of monadic datalog programs via bounded clique-width*

Mikołaj Bojańczyk, Filip Murlak, and Adam Witkowski

University of Warsaw

Abstract. Containment of monadic datalog programs over data trees (labelled trees with an equivalence relation) is undecidable. Recently, decidability was shown for two incomparable fragments: downward programs, which never move up from visited tree nodes, and linear child-only programs, which have at most one intensional predicate per rule and do not use descendant relation. As different as the fragments are, the decidability proofs hinted at an analogy. As it turns out, the common denominator is admitting bounded clique-width counter-examples to containment. This observation immediately leads to stronger decidability results with more elegant proofs, via decidability of monadic second order logic over structures of bounded clique-width. An argument based on two-way alternating tree automata gives a tighter upper bound for linear child-only programs, closing the complexity gap: the problem is 2-ExpTime-complete. As a step towards these goals, complexity of containment over arbitrary structures of bounded clique-width is analysed: satisfiability and containment of monadic programs with stratified negation is in 3-ExpTime, and containment of a linear monadic program in a monadic program is in 2-ExpTime.

1 Introduction

One of the central questions of database theory is that of query containment: deciding if the answers to one query are always contained in the answers to another query, regardless of the content of the database. Being a generalization of satisfiability, containment is undecidable for queries expressed in first order logic (FO), but it is decidable for more restrictive classes of queries like unions of conjunctive queries (UCQs) [6], that is, queries expressible in the positive-existential fragment of FO. A way to go beyond FO without losing decidability, is to add recursion (equivalently, least fixed point operator) to unions of conjunctive queries: the resulting language is datalog. In general, containment of datalog programs is undecidable [19], but it becomes decidable under restriction to monadic datalog programs (equivalently, when the use of least fixed point operator is limited to unary formulae) [7].

In this work we are interested in a particular class of structures, called data trees, which are trees labelled with a finite alphabet with an additional equivalence relation over nodes (modelling data equality). The main motivation for

* Supported by Poland's National Science Centre grant UMO-2013/11/D/ST6/03075.

studying data trees is that they are a convenient model for data organized in a hierarchical structure, for instance, XML documents. When the class of structures is restricted to data trees, containment is still decidable for UCQs [3], but it is undecidable for monadic datalog [1]. A line of research focused on XML applications investigates XPath, an XML query language, which in some variants allows recursion in the form of Kleene star [12,18]. The positive fragment of XPath can be translated to monadic datalog, but the converse translation is not possible due to XPath's limited abilities of testing data equality. In pure datalog setting, two natural fragments were recently shown decidable: downward programs, which never move up from visited tree nodes (cf. [12]), and linear child-only programs, which do not use descendant relation and do not branch (i.e., have at most one intensional predicate per rule) [17]. Relying on ad-hoc arguments, [17] sheds little light on the real reasons behind the decidability, and may give an impression that decidability of these two seemingly different fragments of datalog is pure coincidence. This work puts these two results in context and finds a common denominator, which leads to cleaner arguments, more general results, and—in some cases—tightened complexity bounds.

We show that the common feature of the two fragments is that they both admit counter-examples to containment of clique-width [10] linear in the size of the programs. This almost immediately gives decidability of containment, because monadic datalog is equivalent to monadic second order logic (MSO) [13], for which satisfiability is decidable over structures of bounded clique-width [9]. Unlike tree-width [14], clique-width has not been investigated in the context of datalog. The reason is that, for fixed k, a tree decomposition of width k can be computed in linear time for graphs of tree-width k [4], but for clique-width the best currently known polynomial time algorithm computes decompositions of width $2^{k+1}-1$ for graphs of clique-width k [16]. Given that algorithms relying on decompositions are typically exponential in k, this results in a double exponential constant, which is impractical most of the time. For the purpose of our work, however, constructing a decomposition for a given structure is not an issue: we need to test if there *exists* a decomposition that yields some counter-example. A closer look at the MSO based approach gives a 3-ExpTime upper bound; for linear programs we provide a more economic construction, which gives 2-ExpTime upper bound (even for containment in arbitrary monadic programs).

This approach does not guarantee optimal complexity: for downward programs containment is 2-ExpTime-complete, and ExpSpace-complete under restriction to linear programs [17]. But in some cases it actually tightens the bounds: for linear child-only programs the complexity bounds were 2-ExpTime-hard and in 3-ExpTime, and our method gives a 2-ExpTime algorithm, thus closing the complexity gap. Also, the classes of programs for which the algorithms work are broader; for instance, we can test containment in arbitrary monadic programs, not just downward, or linear child-only.

The paper is organized as follows. In Section 2 we recall basic definitions. In Section 3 we focus on datalog over (arbitrary) structures of bounded clique width. We show that a datalog program with stratified negation can be translated into a triple exponential tree automaton working over clique-width decompositions; this implies that satisfiability and containment of such programs over structures of bounded clique-width is in 3-ExpTime. For *linear* monadic programs without negation we provide a construction going via two-way alternating automata [7,21], which gives a 2-ExpTime upper bound for containment (even in arbitrary monadic programs). In Section 4 we apply these results to the problem of containment over data trees for downward programs and linear child-only programs. In Section 5 we conclude with a brief discussion of the obtained results.

2 Preliminaries

Finite structures and clique-width. Let $\tau = \{R_1, \ldots, R_\ell\}$ be a relational signature, i.e., a set of predicate symbols with arities $ar(R_i)$. A (finite) τ-structure $\mathbb{A}$ is a tuple $\langle A, R_1^{\mathbb{A}}, \ldots, R_\ell^{\mathbb{A}} \rangle$ consisting of universe A and relations $R_i^{\mathbb{A}} \subseteq A^{\mathrm{ar}(R_i)}$ (interpretations of the predicates). A k-coloured τ-structure is a pair $(\mathbb{A}, \gamma)$, consisting of a τ-structure $\mathbb{A}$ and a mapping $\gamma : A \to \{1, \ldots, k\}$, assigning colours to elements of the universe of $\mathbb{A}$.

Clique width of structures is defined by means of an appropriate notion of decomposition, traditionally known as *k-expression (over τ)*. It is defined as a term over the following set of operations (function symbols) $\mathrm{Op}(\tau, k)$:

- $\mathrm{new}(i)$ for $1 \leq i \leq k$, nullary,
- $\rho(i, j)$ for $1 \leq i, j \leq k$, unary,
- $R(i_1, \ldots, i_r)$ for predicates $R \in \tau$ of arity r and $1 \leq i_1, \ldots, i_r \leq k$, unary,
- $\oplus$, binary.

With k-expression e we associate a k-coloured τ-structure $[\![e]\!]$:

- $[\![\mathrm{new}(i)]\!]$ is a structure with a single element, coloured i, and empty relations;
- $[\![\rho(i, j)(e)]\!]$ is obtained from $[\![e]\!]$ by recolouring all elements of colour i to j;
- $[\![R(i_1, \ldots, i_r)(e)]\!]$ is obtained from $[\![e]\!] = (\mathbb{A}, \gamma)$ by adding to $R^{[\![e]\!]}$ all tuples $(a_1, \ldots, a_r)$ such that $a_j \in A$ and $\gamma(a_j) = i_j$ for $1 \leq j \leq r$;
- $[\![e \oplus e']\!]$ is the disjoint union of $[\![e]\!]$ and $[\![e']\!]$.

A *k-expression for* $\mathbb{A}$ is any k-expression e such that $[\![e]\!] = (\mathbb{A}, \gamma)$ for some γ. The *clique-width* of $\mathbb{A}$ is the least k such that there exists a k-expression for $\mathbb{A}$.

Datalog. We assume some familiarity with datalog and only briefly recall its syntax and semantics; for more details see [2] or [5].

A *datalog program* $\mathcal{P}$ over a relational signature σ, split into *extensional* predicates σ_{ext} and *intensional* predicates σ_{int}, is a finite set of rules of the form *head* $\leftarrow$ *body*, where *head* is an atom over σ_{int} and *body* is a (possibly empty) conjunction of atoms over σ written as a comma-separated list. All variables in the body that are not used in the head are implicitly quantified existentially.

Program $\mathcal{P}$ is evaluated on σ_{ext}-structure $\mathbb{A}$ by generating all atoms over σ_{int} that can be inferred from $\mathbb{A}$ by applying the rules repeatedly, to the point of saturation. Each inferred atom can be witnessed by a *proof tree*: an atom inferred by a rule r from intensional atoms $A_1, A_2, \dots, A_n$ (and some extensional atoms) is witnessed by a proof tree whose root has label r and n children which are the roots of the proof trees for atoms A_i (if r has no intensional predicates in its body then the root has no children). The program returns set $\mathcal{P}(\mathbb{A})$ consisting of those inferred atoms that match a distinguished *goal predicate* G.

In programs with *stratified negation* we assume that signature σ is partitioned into strata $\sigma_{\text{ext}} = \sigma_0, \sigma_1, \dots, \sigma_{n-1}, \sigma_n = \{G\}$ for some $n \in \mathbb{N}$. For each $i > 0$, rules for predicates from stratum σ_i contain atoms over $\sigma_0 \cup \dots \cup \sigma_i$ and negated atoms over predicates from $\sigma_0 \cup \dots \cup \sigma_{i-1}$. The partition of σ induces a partition of $\mathcal{P}$ into $\mathcal{P}_1, \dots, \mathcal{P}_n$. The evaluation is done stratum by stratum, that is $\mathcal{P}_i$ is run over atoms inferred by strata $\mathcal{P}_1, \dots, \mathcal{P}_{i-1}$, including those coming directly from the structure.

In this paper we consider only *monadic* programs, i.e., programs whose intensional predicates are at most unary. A datalog program is *linear*, if the right-hand side of each rule contains at most one atom with an intensional predicate (proof trees for such programs are single branches, and we call them proof words).

For programs $\mathcal{P}, \mathcal{Q}$ with a common goal predicate G, we say that program $\mathcal{P}$ is *contained* in program $\mathcal{Q}$, written as $\mathcal{P} \subseteq \mathcal{Q}$, if

$$\mathcal{P}(\mathbb{A}) \subseteq \mathcal{Q}(\mathbb{A})$$

for each σ_{ext}-structure $\mathbb{A}$. Note that if goal predicate G is nullary, this means that if $G \in \mathcal{P}(t)$ then $G \in \mathcal{Q}(t)$; that is, if $\mathcal{P}$ says *true*, so does $\mathcal{Q}$.

Automata. A *ranked alphabet* Γ is a set of letters with arities. A *tree over ranked alphabet* Γ is an ordered tree labelled with elements of Γ such that the number of children of any given node is equal to the arity of its label. Trees over ranked alphabet Γ can be seen as terms over Γ, and vice versa. A term of the form $f(t_1, t_2, \dots, t_n)$, $n = \text{ar}(f)$ corresponds to a tree whose root has label f, and children $v_1, \dots, v_n$ where the subtree rooted at v_i corresponds to the term t_i. Thus, k-expressions over τ are trees over ranked alphabet $\text{Op}(\tau, k)$. An *unranked tree* is a tree without any constraints on the number of children.

A *two-way alternating tree automaton (2ATA)* $\mathcal{A} = \langle \Gamma, Q, q_I, \delta \rangle$ consists of a ranked alphabet Γ, a finite set of states Q, an initial state $q_I \in Q$, and a transition function

$$\delta \colon Q \times \Gamma \to \text{BC}^+\big(Q \times \mathbb{Z}\big)$$

describing actions of automaton $\mathcal{A}$ in state q in a node with label f as a positive Boolean combination of atomic actions of the form (q, d), where $-1 \le d \le \text{ar} f$.

A *run* r of $\mathcal{A}$ over tree t is an unranked tree labelled with pairs (q, v), where q is a state of $\mathcal{A}$ and v is a node of t, satisfying the following condition: if a node of r with label (q, v) has children with labels $(q_1, v_1), \dots, (q_n, v_n)$, and v has label f in t, then there exist $d_1, \dots, d_n \in \mathbb{N}$ such that

- v_i is the d_i'th child of v in t for all i such that $d_i > 0$;
- $v_i = v$ for all i such that $d_i = 0$;
- v_i is the parent of v in t for all i such that $d_i = -1$; and

– Boolean combination $\delta(q, f)$ evaluates to *true* when atomic actions $(q_1, d_1), \ldots, (q_n, d_n)$ are substituted by *true*, and other atomic actions are substituted by *false*.

Tree t is *accepted* by automaton $\mathcal{A}$ if it admits a finite run. By $L(\mathcal{A})$ we denote the language *recognized* by $\mathcal{A}$; that is, the set of trees accepted by $\mathcal{A}$.

A *nondeterministic (one-way) tree automaton (NTA)* is an alternating two-way automaton such that each $\delta(q, f)$ is a disjunction of expressions of the from $(q_1, 1) \wedge (q_2, 2) \wedge \cdots \wedge (q_{\mathrm{ar}f}, \mathrm{ar}f)$.

3 Evaluating monadic datalog over k-expressions

It is a part of the database theory folklore that every monadic datalog program can be translated to a formula of monadic second order logic (MSO) [13]. For concreteness, let us assume the syntax of MSO formulae over signature τ is

$$\begin{aligned}\varphi, \psi ::= \forall X\, \varphi \,|\, \exists X\, \varphi \,|\, \varphi \wedge \psi \,|\, \varphi \vee \psi \,|\, \varphi \to \psi \,|\, \varphi \leftrightarrow \psi \,|\, \neg\varphi \,|\\ |\, X \subseteq Y \,|\, X = \emptyset \,|\, \mathit{singleton}(X) \,|\, R(X_1, \ldots, X_r)\end{aligned}$$

for $R \in \tau$, $r = \mathrm{ar}R$; the semantics is as usual. As is also well known, each MSO formula (over arbitrary structures) can be translated to an equivalent formula over k-expressions (see e.g. [15, Lemma 16]). Finally, each MSO formula over trees can be translated to an equivalent tree automaton [11,20]. Thus, evaluating a monadic Datalog program over a structure reduces to running an appropriate nondeterministic automaton over any k-expression for this structure. With a bit of care we can ensure that the automaton does not grow too fast.

Proposition 1. *Let k be a positive integer and $\mathcal{P}$ a monadic datalog program with stratified negation. One can construct (in time polynomial in the size of the output) a triple exponential NTA $\mathcal{A}_\mathcal{P}$ recognizing k-expressions e such that $[\![e]\!] = (\mathbb{A}, \gamma)$ and $\mathcal{P}(\mathbb{A}) \neq \emptyset$.*

Moreover, if $\mathcal{P}$ uses no negation, one can construct (in time polynomial in the size of the output) a double exponential NTA $\mathcal{A}_{\neg\mathcal{P}}$ recognizing k-expressions e such that $[\![e]\!] = (\mathbb{A}, \gamma)$ and $\mathcal{P}(\mathbb{A}) = \emptyset$.

Proof. Program $\mathcal{P}$ with p intensional predicates and at most q variables per rule can be translated to a linear-size MSO formula φ of the form

$$\forall X_1 \ldots \forall X_p\, \exists X_{p+1} \ldots \exists X_{p+q}\, \varphi_0(X_1, \ldots, X_{p+q})$$

where φ_0 is a quantifier-free formula over signature σ_{ext}, such that $\mathbb{A} \models \varphi$ if and only if $\mathcal{P}(\mathbb{A}) \neq \emptyset$ [13]. For programs with stratified negation we do not need to introduce arbitrary number of alternations. Without loss of generality we can assume that intensional predicates are split into positive, used only under even number of negations, and negative, used only under odd number of negations. One can obtain a linear-size formula of the form

$$\exists X_1 \ldots \exists X_\ell\, \forall X_{\ell+1} \ldots \forall X_m \exists X_{m+1} \ldots \exists X_n\, \varphi_0(X_1, \ldots, X_n)$$

where, roughly speaking, the first block of quantifiers introduces the negative predicates, the second deals with closure properties for the negative predicates and introduces the positive predicates, and the third deals with closure properties of the positive predicates.

The next step is to translate φ to a formula $\hat{\varphi}$ over the signature of k-expressions over σ_{ext}, such that for every σ_{ext}-structure $\mathbb{A}$ and every k-expression e for $\mathbb{A}$, $\mathbb{A} \models \varphi$ iff $e \models \hat{\varphi}$. We follow the translation from [15, Lemma 16]. It relies on the assumption that the universe of structure $\mathbb{A}$ is contained in the set of nodes of k-expression e: each node with label $\text{new}(i)$ is identified with the element of $\mathbb{A}$ it represents. The translation does two things. It relativises the quantifiers to the set of leaves; that is, it replaces φ_0 with

$$\bigwedge_{i=1}^{\ell} \mathit{leaf}(X_i) \wedge \left(\bigwedge_{i=\ell+1}^{m} \mathit{leaf}(X_i) \rightarrow \bigwedge_{i=m+1}^{n} \mathit{leaf}(X_i) \wedge \varphi_0(X_1, \ldots, X_n) \right)$$

where $\mathit{leaf}(X_j)$ is an auxiliary formula saying that each element of X_j is a leaf. Then, it replaces each atomic formula $R(X_{j_1}, \ldots, X_{j_r})$ in φ_0 with formula

$$\psi_R(X_{j_1}, \ldots, X_{j_r})$$

saying that there is a node v with label $R(i_1, \ldots, i_r)$ such that some leaves $x_1 \in X_{j_1}, \ldots, x_r \in X_{j_r}$ are descendants of v and have colours $i_1, \ldots, i_r$ according to the current colouring in v. Note that the obtained formula $\hat{\varphi}$ only uses tree relations (child and labels) in the auxiliary formulae *leaf* and ψ_R for $R \in \sigma_{\text{ext}}$, which, incidentally, are not quantifier free. Instead of expressing these formulae in MSO, we shall keep them as primitives, to be translated directly to automata.

Translation of MSO formulae to automata [11,20] is done by induction over the structure of the formula: for each quantifier free subformula $\eta(X_{j_1}, \ldots, X_{j_\ell})$ of $\hat{\varphi}$ we construct a deterministic automaton over alphabet $\{0,1\}^n \times \text{Op}(\sigma_{\text{ext}}, k)$ that accepts tree t if and only if

$$(t', U_1, \ldots, U_\ell) \models \eta(X_{j_1}, \ldots, X_{j_\ell})$$

where tree t' is obtained from t by projecting the labels to $\text{Op}(\sigma_{\text{ext}}, k)$, and $U_1, \ldots, U_\ell$ are the sets of nodes whose label in t has 1 in coordinates $j_1, \ldots, j_\ell$, respectively. For subformulae $X_{j_1} \subseteq X_{j_2}$, $\mathit{singleton}(X_j)$, $X_j = \emptyset$, and $\mathit{leaf}(X_j)$, there are automata of constant-size state-space (though over exponential alphabet). For $\psi_R(X_{j_1}, \ldots, X_{j_r})$ the automaton has $2^{kr}+1$ states: it works bottom-up maintaining sets $I_1, \ldots, I_r$ of colours assigned to elements of $X_{j_1}, \ldots, X_{j_r}$ by the colouring corresponding to the current node; it accepts if at any moment it finds a node with label $R(i_1, \ldots, i_r)$ for some $i_1 \in I_1, \ldots, i_r \in I_r$. The boolean connectives are realized by an appropriate product construction (negation is straightforward for deterministic automata). Altogether we end up with a product of linearly many deterministic automata of size at most single exponential; this gives a single exponential deterministic automaton $\mathcal{A}$ for the quantifier-free part of formula $\hat{\varphi}$. Automaton $\mathcal{A}_{\mathcal{P}}$ can be obtained from $\mathcal{A}$ by projecting out coordinates $m+1, \ldots, n$ of the labels, complementing, projecting out coordinates

$\ell+1,\dots,m$, complementing, and projecting out coordinates $1,\dots,\ell$. Since each complementation involves exponential blow-up, the resulting automaton is triple exponential in the size of the program.

Automaton $\mathcal{A}_{\neg\mathcal{P}}$ is obtained from the negation of the first MSO formula in this proof; it requires only one complementation of a nondeterministic automaton, resulting in a double exponential bound. □

As an immediate corollary we get that satisfiability in structures of bounded clique-width can be tested in 3-ExpTime for monadic programs with stratified negation. The following is a special case of this.

Corollary 1. *Given monadic programs $\mathcal{P},\mathcal{Q}$ and $k \in \mathbb{N}$, one can decide in* 3-ExpTime *if $\mathcal{P}(\mathbb{A}) \subseteq \mathcal{Q}(\mathbb{A})$ for all structures $\mathbb{A}$ of clique-width at most k.*

If the "smaller" program is linear, we get better complexity. Our main technical contribution here is a direct translation from linear monadic datalog to 2ATA. Unlike in [7], were 2ATA worked on proof trees and essentially mimicked behaviour of datalog programs, we work on k-expressions, in which distant leaves may represent nodes that are in fact close together. The main idea is that each time the 2ATA sees $\oplus$, it guesses a way to cut the proof word into subwords to be realised in the left and right subtree (see appendix for the details).

Theorem 1. *For a linear monadic program $\mathcal{P}$ and $k \in \mathbb{N}$ one can construct (in time polynomial in the size of the output) a single exponential 2ATA $\mathcal{A}$ recognizing k-expressions e such that $[\![e]\!] = (\mathbb{A},\gamma)$ and $\mathcal{P}(\mathbb{A}) \neq \emptyset$.*

We shall see later that these bounds are tight in the sense that obtaining a polynomial bound would violate lower bounds on the containment problem over data trees discussed in the following section.

The second, and last, step of the construction relies on the following theorem.

Theorem 2 ([7,21]). *For a given 2ATA $\mathcal{A}$ one can construct (in time polynomial in the size of the output) single exponential NTA $\mathcal{B}$ recognizing $L(\mathcal{A})$.*

Combining Theorem 1, Theorem 2, and the additional claim of Proposition 1, we obtain the following bounds.

Corollary 2. *Given a linear monadic program $\mathcal{P}$, a monadic program $\mathcal{Q}$, and $k \in \mathbb{N}$, one can decide in* 2-ExpTime *if $\mathcal{P}(\mathbb{A}) \subseteq \mathcal{Q}(\mathbb{A})$ for all structures $\mathbb{A}$ of clique-width at most k.*

Proof. If the goal predicate G of $\mathcal{P}$ and $\mathcal{Q}$ is nullary, the claim follows immediately: one constructs an NTA $\mathcal{A}_{\mathcal{P}}$ recognizing the set of k-expressions e such that $[\![e]\!] = (\mathbb{A},\gamma)$ and $G \in \mathcal{P}(\mathbb{A})$, and an NTA $\mathcal{A}_{\neg\mathcal{Q}}$ recognizing the set of k-expressions e such that $[\![e]\!] = (\mathbb{A},\gamma)$ and $G \notin \mathcal{P}(\mathbb{A})$, and checks if $L(\mathcal{A}_{\mathcal{P}}) \cap L(\mathcal{A}_{\neg\mathcal{Q}}) \neq \emptyset$.

Assume that G is unary. Extend σ_{ext} with a fresh unary relation H. Let $\mathcal{P}_0$ be obtained from $\mathcal{P}$ by adding rule $G_0 \leftarrow G(x), H(x)$ for a fresh nullary predicate G_0 and changing the goal predicate to G_0; similarly construct $\mathcal{Q}_0$ from $\mathcal{Q}$. Now, it is enough to check if $L(\mathcal{A}_{\mathcal{P}_0}) \cap L(\mathcal{A}_{\neg\mathcal{Q}_0}) \cap L(\mathcal{B}) \neq \emptyset$, where $\mathcal{B}$ is an automaton recognizing the set of k-expressions e over signature $\sigma_{\text{ext}} \cup \{H\}$ such that $[\![e]\!] = (\mathbb{A},\gamma)$ and $H^{\mathbb{A}}$ is a singleton. □

4 Containment over data trees

We now restrict the class of structures to *data trees*; that is, (finite) labelled unranked trees over $\Gamma \times \mathsf{DVal}$, where Γ is a finite alphabet and DVal is an infinite set of so-called *data values*. Datalog programs over data trees refer to relations: child $\downarrow$, descendant $\downarrow^+$, data value equality $\sim$, and label tests $a \in \Gamma$. That is, a data tree is seen as a relational structure over signature τ_{dt} consisting of binary relations $\{\downarrow, \downarrow^+, \sim\}$ and unary relations Γ.

We are interested in the problem of containment over data trees: we say that *program $\mathcal{P}$ is contained in program $\mathcal{Q}$ over data trees*, written as $\mathcal{P} \subseteq_{\mathrm{dt}} \mathcal{Q}$, if

$$\mathcal{P}(t) \subseteq \mathcal{Q}(t)$$

for each data tree t. In this section by containment we always mean containment over data trees. Thus, the containment problem is: given programs $\mathcal{P}$, $\mathcal{Q}$ over data trees, decide if $\mathcal{P} \subseteq_{\mathrm{dt}} \mathcal{Q}$.

We propose the following generic approach:

1. show that containment over all data trees is equivalent to containment over data trees of clique-width at most k;
2. test containment over k-expressions for data-trees.

The second step is easy for arbitrary monadic programs (Section 4.1). Hence, given that containment over data trees is undecidable for monadic programs [1], the first step can only be carried out for restricted fragments of monadic datalog. In what follows we shall consider two such fragments: downward programs (Section 4.2), and linear child-only programs (Section 4.3).

4.1 Containment over data trees of bounded clique-width

In the light of the general results of the previous section, testing containment over data trees of clique-width at most k is almost straightforward: the only issue is that not all k-expressions over signature τ_{dt} yield data-trees. But those that do can be recognized by a tree automaton. Since the bound on the clique-width depends on the size of the program, the construction requires some care.

Lemma 1. *For all k, k-expressions for data trees form a regular language. The size of automaton recognizing those k-expressions is double exponential in k.*

Proof. To prove the claim, it suffices to give a double-exponential bottom-up automaton that given a k-expression for structure $\mathbb{A}$ verifies that:

- every element is reachable from root via directed $\downarrow$-path;
- the relation $\downarrow \cup \downarrow^{-1}$ is acyclic;
- $\downarrow^+$ is transitive closure of $\downarrow$;
- $\sim$ is reflexive, symmetric and transitive.

The construction is straightforward. The details are given in the appendix. □

This suffices to solve containment over data trees of bounded clique-width.

Proposition 2. *Given monadic programs $\mathcal{P}, \mathcal{Q}$ and $k \in \mathbb{N}$, one can test in* 3-ExpTime *whether $\mathcal{P}(t) \subseteq \mathcal{Q}(t)$ for every data tree t of clique-width at most k. If $\mathcal{P}$ is linear, the complexity drops to* 2-ExpTime.

Proof. The proofs are just like for Corollary 1 and Corollary 2, except that the automaton recognizing counter-examples to containment has to be intersected with the automaton recognizing k-expressions that yield data trees. □

4.2 Downward programs

As we have explained, showing that containment over data trees of two programs is equivalent to containment over data trees of bounded clique-width immediately gives decidability of containment in 3-ExpTime in general, and in 2-ExpTime if the "smaller" problem is linear. In this section we apply this method to the class of downward programs. We do it mainly for illustrative purposes, as it is known that for downward programs containment is 2-ExpTime-complete in general, and ExpSpace-complete for linear programs [17]. But our method also gives broader results: it uses a relaxed definition of downward programs, and it works for testing containment in arbitrary monadic programs.

We begin with an observation that has been seminal to this work. Let $\mathit{datacut}(t)$ be the maximum over all $\downarrow$-edges (u, v) in t of the number of $\sim$-classes represented in both parts of t: the subtree rooted at v, and the rest of the tree.

Lemma 2. *Every data tree t has clique-width at most $4 \cdot \mathit{datacut}(t) + 5$.*

Proof. Let $k = 4 \cdot \mathit{datacut}(t) + 5$. We will use colours of the form

$$\{\mathit{root}, \mathit{notroot}\} \times \{0, 1, \dots, \mathit{datacut}(d)\} \times \{\mathit{old}, \mathit{new}\}$$

plus an additional colour *temp*. We construct a k-expression inductively for subforests f of t, maintaining the following invariants for the induced colouring

1. all nodes have colours (x, y, old),
2. colours (root, y, z) are reserved for the roots of f,
3. colours $(\mathit{notroot}, y, z)$ are used by other nodes,
4. node has colour $(x, 0, z)$ iff it carries a data value never used outside of f.

Colors (x, y, new) are used when combining two parts of the tree with $\oplus$, to avoid gluing colours together. Color *temp* is used for recolouring.

For single-node tree s, we just use $(\mathit{root}, 1, \mathit{old})$ or $(\mathit{root}, 0, \mathit{old})$, in accordance with invariant 4.

Assume we have k-expressions for all immediate subtrees of subtree s, say $e_1, \dots, e_\ell$. We use $\oplus$ to add them one by one, adding necessary $\sim$-edges. Each time we add another e_i, we first recolour (x, y, old) to (x, y, new) in e_i for all x, y. We add $\sim$-edges as required, and recolour the nodes (using *temp*) to restore invariants 1 and 4.

Next, we want to build a k-expression for s. We use $\oplus$ to add the root, coloured $(\mathit{root}, y, \mathit{new})$, where y is chosen depending on $\sim$ relation between the root and the nodes in the immediate subtrees. We add $\downarrow$ edges between $(\mathit{root}, y, \mathit{new})$ and $(\mathit{root}, y', \mathit{old})$ and $\downarrow^+$ edges between $(\mathit{root}, y, \mathit{new})$ and (x', y', old) for all x', y', as well as appropriate $\sim$-edges. Next, we recolour $(\mathit{root}, y', \mathit{old})$ to $(\mathit{notroot}, y', \mathit{old})$ for all y', and $(\mathit{root}, y, \mathit{new})$ to $(\mathit{root}, y, \mathit{old})$. If necessary, we do additional recolouring to restore invariant 4. □

A datalog rule is essentially a conjunctive query, so one can associate with it a relational structure $\mathbb{A}_r$ in the usual way: the universe is the set $\mathrm{var}(r)$ of variables used in r, and relations are defined by extensional atoms of r. Recall that program $\mathcal{P}$ is *downward* if it had no nullary predicates and for each rule $r \in \mathcal{P}$, graph (V, E) with $V = \mathrm{var}(r)$ and $E = (\downarrow)^{\mathbb{A}_r} \cup (\downarrow^+)^{\mathbb{A}_r}$ was a tree in which the variable used in the head of r is the root [17]. Here we use a relaxed variant of this definition: we lift the restriction that the graph is a tree, but we keep the requirement that each node is reachable from the variable used in the head.

Theorem 3. *Let $\mathcal{P}$ be a downward program and $\mathcal{Q}$ an arbitrary monadic program. If $\mathcal{P} \not\subseteq \mathcal{Q}$, then there exists a witness with clique-width at most $4 \cdot \|\mathcal{P}\| + 5$.*

Proof. The theorem follows immediately from the following claim (with A set to the goal predicate G): for each tree t and each atom $A(v)$ inferred by $\mathcal{P}$ from t, there exists a data tree t' such that

- $\mathit{datacut}(t') \leq \|\mathcal{P}\|$,
- $A(\mathit{root}_{t'})$ can be inferred by $\mathcal{P}$ from t', where $\mathit{root}_{t'}$ is the root of t',
- there exists a homomorphism from t' to t that maps $\mathit{root}_{t'}$ to v (in particular, $G(\mathit{root}_{t'}) \notin \mathcal{Q}(t')$ unless $G(v) \in \mathcal{Q}(t)$).

We prove the claim by induction on the size of proof tree p witnessing $A(v)$. Let r be the rule in the root of p and let $h \colon \mathbb{A}_r \to t$ be the associated homomorphism (it maps the head variable to v). Let t_r be the data tree obtained from t_v (the subtree of t rooted at v) by interpreting $\sim$ as the least equivalence relation extending the image of $\sim^{\mathbb{A}_r}$ under h; it has at most $|r|$ non-singleton abstraction classes. Let $R_1(y_1), \ldots, R_m(y_m)$ be the intensional atoms in rule r. By the inductive hypothesis there exist appropriate $t_1, \ldots, t_m$ for $R_1(h(y_1)), \ldots, R_m(h(y_m))$. We obtain t' by taking disjoint union of t_r and $t_1, \ldots, t_m$ with the roots of $t_1, \ldots, t_m$ identified with $h(y_1), \ldots, h(y_m)$, and closing $\sim$ and $\downarrow^+$ under transitivity. □

Corollary 3. *Containment of a downward program in a monadic program is in* 3-ExpTime, *and in* 2-ExpTime *if the downward program is also linear.*

4.3 Linear child-only programs

In this section we show a more useful application of results from Section 3. Recall that by child-only programs we mean programs that use only $\downarrow$ and $\sim$ relation, while $\downarrow^+$ is forbidden. It is known that containment for linear child-only programs is in 3-ExpTime and 2-ExpTime-hard (for non-linear ones containment is undecidable) [17]. Here, we close this complexity gap and also extend the result to containment of linear child-only programs in arbitrary monadic ones. Similarly to the downward programs, we show that containment can be verified over data trees of bounded clique-width.

Theorem 4. *For a linear child-only program $\mathcal{P}$ and arbitrary monadic program $\mathcal{Q}$, if $\mathcal{P} \not\subseteq \mathcal{Q}$, there exists a witness with clique-width at most* $8\|\mathcal{P}\|^2 + 3\|\mathcal{P}\| + 2$.

Because child-only programs can go up and down in the tree, it is not possible to bound *datacut* as we did for downward programs. It is possible, however, to bound the number of $\sim$-classes represented at the same time in an appropriately generalized subtree of a node v and in the rest of the tree. The proof of this fact is not very hard but it requires some technical definitions from [17] and is given in the appendix.

Just like for downward programs, the following is a straightforward application of Theorems 1 and 4 (and 2-ExpTime-hardness shown in [17]).

Corollary 4. *Containment of a linear child-only program in a monadic program is 2-*ExpTime*-complete.*

5 Conclusions

We have shown that over structures of bounded clique-width (arbitrary structures and data trees), containment of monadic datalog programs is decidable in 3-ExpTime, and containment of linear monadic programs in monadic programs is in 2-ExpTime. Consequently, decidability of containment for a fragment of monadic datalog reduces to showing that the fragment admits bounded clique-width counter-examples to containment. Graph decompositions have been used before for deciding properties of datalog programs: already in early 1990s Courcelle noticed a connection between runs of datalog programs and tree decompositions of structures, and concluded decidability of some properties of programs expressible in MSO over these decompositions [8]. Over data trees, however, tree-width is not useful: there is much less freedom in constructing models of a program, and neither $\downarrow^+$ nor $\sim$ are sparse relations, as is required by bounded tree-width. Clique-width seems to be exactly the notion that is needed.

We applied this method to generalize previously known decidability results for downward programs and linear child-only programs: we relaxed the definition of downward programs and, more importantly, we covered containment in arbitrary monadic programs. With a bit of extra effort one could further relax the notion of downward programs to allow disconnected rules. More interestingly, there is a relatively natural unifying fragment with decidable containment: linear child-only programs that use freely predicates defined by downward programs. The method seems flexible enough for further extensions. More generally, one could try to port the method to formalisms other then monadic datalog, e.g., XPath, both for data trees and data graphs.

References

1. Serge Abiteboul, Pierre Bourhis, Anca Muscholl, and Zhilin Wu. Recursive queries on trees and data trees. In *ICDT'13*, pages 93–104, 2013.
2. Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases.* Addison Wesley, 1995.
3. Henrik Björklund, Wim Martens, and Thomas Schwentick. Optimizing conjunctive queries over trees using schema information. In *MFCS'08*, pages 132–143, 2008.
4. Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.
5. Stefano Ceri, Georg Gottlob, and Letizia Tanca. *Logic programming and databases.* Springer-Verlag New York, Inc., 1990.
6. Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *STOC'77*, pages 77–90, New York, NY, USA, 1977. ACM.
7. Stavros S. Cosmadakis, Haim Gaifman, Paris C. Kanellakis, and Moshe Y. Vardi. Decidable optimization problems for database logic programs (preliminary report). In *STOC'88*, pages 477–490, 1988.
8. Bruno Courcelle. Recursive queries and context-free graph grammars. *Theor. Comput. Sci.*, 78(1):217–244, 1991.
9. Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000.
10. Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1-3):77–114, 2000.
11. John Doner. Tree acceptors and some of their applications. *J. Comput. Syst. Sci.*, 4(5):406–451, 1970.
12. Diego Figueira. Satisfiability of downward XPath with data equality tests. In *PODS'09*, pages 197–206, 2009.
13. Georg Gottlob and Christoph Koch. Monadic datalog and the expressive power of languages for web information extraction. *J. ACM*, 51(1):74–113, 2004.
14. Georg Gottlob, Reinhard Pichler, and Fang Wei. Monadic datalog over finite structures of bounded treewidth. *ACM Trans. Comput. Log.*, 12(1):3, 2010.
15. Martin Grohe and György Turán. Learnability and definability in trees and similar structures. *Theory Comput. Syst.*, 37(1):193–220, 2004.
16. Petr Hlinený and Sang-il Oum. Finding branch-decompositions and rank-decompositions. *SIAM J. Comput.*, 38(3):1012–1032, 2008.
17. Filip Mazowiecki, Filip Murlak, and Adam Witkowski. Monadic datalog and regular tree pattern queries. In *MFCS'14*, pages 426–437, 2014.
18. Frank Neven and Thomas Schwentick. On the complexity of XPath containment in the presence of disjunction, DTDs, and variables. *Logical Methods in Computer Science*, 2(3), 2006.
19. Oded Shmueli. Equivalence of datalog queries is undecidable. *J. Log. Program.*, 15(3):231–241, 1993.
20. James W. Thatcher and Jesse 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.
21. Moshe Y. Vardi. Reasoning about the past with two-way automata. In *ICALP'98*, pages 628–641, 1998.