Monadic Datalog Containment on Trees

André Frochaux[1], Martin Grohe[2], and Nicole Schweikardt[1]

[1] Goethe-Universität Frankfurt am Main,
`{afrochaux,schweika}@informatik.uni-frankfurt.de`
[2] RWTH Aachen University, `grohe@informatik.rwth-aachen.de`

Abstract. We show that the query containment problem for monadic datalog on finite unranked labeled trees can be solved in 2-fold exponential time when (a) considering unordered trees using the axes *child* and *descendant*, and when (b) considering ordered trees using the axes *firstchild*, *nextsibling*, *child*, and *descendant*. When omitting the *descendant*-axis, we obtain that in both cases the problem is EXPTIME-complete.

1 Introduction

The query containment problem (QCP) is a fundamental problem that has been studied for various query languages. Datalog is a standard tool for expressing queries with recursion. From Cosmadakis et al. [5] and Benedikt et al. [2] it is known that the QCP for *monadic* datalog queries on the class of all finite relational structures is 2EXPTIME-complete. Restricting attention to finite unranked labeled trees, Gottlob and Koch [8] showed that on *ordered* trees the QCP for monadic datalog is EXPTIME-hard and decidable, leaving open the question of a tight bound.

Here we show a matching EXPTIME upper bound for the QCP for monadic datalog on ordered trees using the axes *firstchild*, *nextsibling*, and *child*. When adding the *descendant*-axis, we obtain a 2EXPTIME upper bound. This, in particular, also yields a 2EXPTIME upper bound for the QCP for monadic datalog on *unordered* trees using the axes *child* and *descendant*, and an EXPTIME upper bound for unordered trees using only the *child*-axis. The former result answers a question posed by Abiteboul et al. in [1]. We complement the latter result by a matching lower bound.

The paper is organised as follows. Section 2 fixes the basic notation concerning datalog queries, (unordered and ordered) trees and their representations as logical structures, and summarises basic properties of monadic datalog on trees. Section 3 presents our main results regarding the query containment problem for monadic datalog on trees. Due to space limitations, most technical details had to be deferred to the full version of this paper, available at `http://arxiv.org/abs/1404.0606`.

2 Trees and Monadic Datalog (mDatalog)

Throughout this paper, Σ will always denote a finite non-empty alphabet.
By $\mathbb{N}$ we denote the set of non-negative integers, and we let $\mathbb{N}_{\geqslant 1} := \mathbb{N} \setminus \{0\}$.

Relational Structures. As usual, a *schema* τ consists of a finite number of relation symbols R, each of a fixed *arity* $ar(R) \in \mathbb{N}_{\geqslant 1}$. A *$\tau$-structure* $\mathcal{A}$ consists of a *finite* non-empty set A called the *domain* of $\mathcal{A}$, and a relation $R^{\mathcal{A}} \subseteq A^{ar(R)}$ for each relation symbol $R \in \tau$. It will often be convenient to identify $\mathcal{A}$ with the *set of atomic facts of* $\mathcal{A}$, i.e., the set $atoms(\mathcal{A})$ consisting of all facts $R(a_1, \ldots, a_{ar(r)})$ for all relation symbols $R \in \tau$ and all tuples $(a_1, \ldots, a_{ar(R)}) \in R^{\mathcal{A}}$.

If τ is a schema and ℓ is a list of relation symbols, we write τ^{ℓ} to denote the extension of the schema τ by the relation symbols in ℓ. Furthermore, τ_{Σ} denotes the extension of τ by new unary relation symbols $\mathbf{label}_{\alpha}$, for all $\alpha \in \Sigma$.

Unordered Trees. An *unordered Σ-labeled tree* $T = (V^T, \lambda^T, E^T)$ consists of a finite set V^T of nodes, a function $\lambda^T : V^T \to \Sigma$ assigning to each node v of T a label $\lambda(v) \in \Sigma$, and a set $E^T \subseteq V^T \times V^T$ of directed edges such that the graph (V^T, E^T) is a rooted tree where edges are directed from the root towards the leaves. We represent such a tree T as a relational structure of domain V^T with unary and binary relations: For each label $\alpha \in \Sigma$, $\mathbf{label}_{\alpha}(x)$ expresses that x is a node with label α; $\mathbf{child}(x, y)$ expresses that y is a child of node x; $\mathbf{root}(x)$ expresses that x is the tree's root node; $\mathbf{leaf}(x)$ expresses that x is a leaf; and $\mathbf{desc}(x, y)$ expresses that y is a descendant of x (i.e., y is a child or a grandchild or ... of x). We denote this relational structure representing T by $\mathcal{S}_u(T)$, but when no confusion arises we simply write T instead of $\mathcal{S}_u(T)$.

The queries we consider for unordered trees are allowed to make use of at least the predicates $\mathbf{label}_{\alpha}$ and $\mathbf{child}$. We fix the schema

$$\tau_u := \{\mathbf{child}\}.$$

The representation of unordered Σ-labeled trees as $\tau_{u,\Sigma}$-structures was considered, e.g., in [1].

Ordered Trees. An *ordered* Σ-labeled tree $T = (V^T, \lambda^T, E^T, order^T)$ has the same components as an unordered Σ-labeled tree and, in addition, $order^T$ fixes for each node u of T, a strict linear order of all the children of u in T.

To represent such a tree as a relational structure, we use the same domain and the same predicates as for unordered Σ-labeled trees, along with three further predicates **fc** ("first-child"), **ns** ("next-sibling"), and **ls** ("last sibling"), where $\mathbf{fc}(x, y)$ expresses that y is the first child of node x (w.r.t. the linear order of the children of x induced by $order^T$); $\mathbf{ns}(x, y)$ expresses that y is the right sibling of x (i.e., x and y have the same parent p, and y is the immediate successor of x in the linear order of p's children given by $order^T$); and $\mathbf{ls}(x)$ expresses that x is the rightmost sibling (w.r.t. the linear order of the children of x's parent given by $order^T$). We denote this relational structure representing T by $\mathcal{S}_o(T)$, but when no confusion arises we simply write T instead of $\mathcal{S}_o(T)$.

The queries we consider for ordered trees are allowed to make use of at least the predicates $\mathbf{label}_{\alpha}$, **fc**, and **ns**. We fix the schemas

$$\tau_o := \{\mathbf{fc}, \mathbf{ns}\} \qquad \textit{and} \qquad \tau_{GK} := \tau_o^{\mathbf{root},\mathbf{leaf},\mathbf{ls}}.$$

In [8], Gottlob and Koch represented ordered Σ-labeled trees as $\tau_{GK,\Sigma}$-structures.

Datalog. We assume that the reader is familiar with the syntax and semantics of *datalog* (cf., e.g., [6,8]). Predicates that occur in the head of some rule of a datalog program $\mathcal{P}$ are called *intensional*, whereas predicates that only occur in the body of rules of $\mathcal{P}$ are called *extensional*. By $\mathrm{idb}(\mathcal{P})$ and $\mathrm{edb}(\mathcal{P})$ we denote the sets of intensional and extensional predicates of $\mathcal{P}$, resp. We say that $\mathcal{P}$ *is of schema* τ if $\mathrm{edb}(\mathcal{P}) \subseteq \tau$. We write $\mathcal{T}_{\mathcal{P}}$ to denote the *immediate consequence operator* associated with a datalog program $\mathcal{P}$. Recall that $\mathcal{T}_{\mathcal{P}}$ maps a set C of atomic facts to the set of all atomic facts that are derivable from C by at most one application of the rules of $\mathcal{P}$ (see e.g. [6,8]). The monotonicity of $\mathcal{T}_{\mathcal{P}}$ implies that for each finite set C, the iterated application of $\mathcal{T}_{\mathcal{P}}$ to C leads to a fixed point, denoted by $\mathcal{T}_{\mathcal{P}}^{\omega}(C)$, which is reached after a finite number of iterations.

Monadic datalog queries. A datalog program belongs to *monadic datalog* (mDatalog, for short), if all its *intensional* predicates have arity 1.

A *unary monadic datalog query* of schema τ is a tuple $Q = (\mathcal{P}, P)$ where $\mathcal{P}$ is a monadic datalog program of schema τ and P is an intensional predicate of $\mathcal{P}$. $\mathcal{P}$ and P are called the *program* and the *query predicate* of Q. When evaluated in a finite τ-structure $\mathcal{A}$ that represents a labeled tree T, the query Q results in the unary relation $Q(T) := \{a \in A \,:\, P(a) \in \mathcal{T}_{\mathcal{P}}^{\omega}(\mathit{atoms}(\mathcal{A}))\}$.

The *Boolean monadic datalog query* Q_{Bool} specified by $Q = (\mathcal{P}, P)$ is the Boolean query with $Q_{\mathit{Bool}}(T) = \textbf{yes}$ iff the tree's root node belongs to $Q(T)$.

The *size* $\|Q\|$ of a monadic datalog query Q is the length of $Q = (\mathcal{P}, P)$ viewed as a string over a suitable alphabet.

Expressive power of monadic datalog on trees. From Gottlob and Koch [8] we know that on *ordered* Σ-labeled trees represented as $\tau_{GK,\Sigma}$-structures, monadic datalog can express exactly the same unary queries as monadic second-order logic — for short, we will say "mDatalog(τ_{GK}) = MSO(τ_{GK}) on ordered trees". Since the **child** and **desc** relations are definable in MSO(τ_{GK}), this implies that mDatalog(τ_{GK}) = mDatalog($\tau_{GK}^{\textbf{child},\textbf{desc}}$) on ordered trees.

On the other hand, using the monotonicity of the immediate consequence operator, one obtains that removing any of the predicates $\textbf{root}, \textbf{leaf}, \textbf{ls}$ from τ_{GK} strictly decreases the expressive power of mDatalog on ordered trees (see [7]). By a similar reasoning one also obtains that on *unordered* trees, represented as $\tau_{u,\Sigma}^{\textbf{root},\textbf{leaf},\textbf{desc}}$-structures, monadic datalog is strictly less expressive than monadic second-order logic, and omitting any of the predicates **root**, **leaf** further reduces the expressiveness of monadic datalog on unordered trees [7].

3 Query Containment for Monadic Datalog on Trees

Let τ_{Σ} be one of the schemas introduced in Section 2 for representing (ordered or unordered) Σ-labeled trees as relational structures. For two unary queries Q_1 and Q_2 of schema τ_{Σ} we write $Q_1 \subseteq Q_2$ to indicate that for every Σ-labeled tree T we have $Q_1(T) \subseteq Q_2(T)$. Similarly, if Q_1 and Q_2 are *Boolean* queries of schema τ_{Σ}, we write $Q_1 \subseteq Q_2$ to indicate that for every Σ-labeled tree T, if $Q_1(T) = \textbf{yes}$ then also $Q_2(T) = \textbf{yes}$. We write $Q_1 \not\subseteq Q_2$ to indicate that

$Q_1 \subseteq Q_2$ does not hold. The *query containment problem* (QCP, for short) is defined as follows:

> THE QCP FOR mDatalog(τ) ON TREES
> *Input:* A finite alphabet Σ and
> two (unary or Boolean) mDatalog(τ_Σ)-queries Q_1 and Q_2.
> *Question:* Is $Q_1 \subseteq Q_2$?

It is not difficult to see that this problem is decidable: the first step is to observe that monadic datalog can effectively be embedded into monadic second-order logic, the second step then applies the well-known result that the monadic second-order theory of finite labeled trees is decidable (cf., e.g., [11,4]).

Regarding ordered trees represented as τ_{GK}-structures, in [8] it was shown that the QCP for unary mDatalog(τ_{GK})-queries on trees is EXPTIME-hard. Our first main result generalises this to unordered trees represented as τ_u-structures:

Theorem 1
The QCP for Boolean mDatalog(τ_u) *on unordered trees is* EXPTIME-*hard.*

Our proof proceeds via a reduction from the EXPTIME-complete *two person corridor tiling* (TPCT) problem [3]: For a given instance I of the TPCT-problem we construct (in polynomial time) an alphabet Σ and two Boolean mDatalog($\tau_{u,\Sigma}$)-queries Q_1, Q_2 which enforce that any tree T witnessing that $Q_1 \not\subseteq Q_2$, contains an encoding of a winning strategy for the first player of the TPCT-game associated with I. Using Theorem 1 along with a method of [8] for replacing the **child**-predicate by means of the predicates **fc**, **ns**, we can transfer the hardness result to ordered trees represented by τ_o-structures:

Corollary 2
The QCP for Boolean mDatalog(τ_o) *on ordered trees is* EXPTIME-*hard.*

Our second main result provides a matching EXPTIME upper bound for the QCP on ordered trees, even in the presence of all predicates in $\tau_{GK}^{\mathbf{child}}$:

Theorem 3
The QCP for unary mDatalog($\tau_{GK}^{\mathbf{child}}$) *on ordered trees belongs to* EXPTIME.

Proof (sketch). Consider a schema $\tau \subseteq \tau_{GK}^{\mathbf{child},\mathbf{desc}}$. By using the automata-theoretic approach [5], a canonical method for deciding the QCP for unary mDatalog(τ) proceeds as follows:

(1) Transform the input queries Q_1 and Q_2 into *Boolean* queries Q_1' and Q_2' on *binary* trees, such that $Q_1 \subseteq Q_2$ iff $Q_1' \subseteq Q_2'$.
(2) Construct tree automata $\mathtt{A}_1^{\mathbf{yes}}$ and $\mathtt{A}_2^{\mathbf{no}}$ such that $\mathtt{A}_1^{\mathbf{yes}}$ (resp. $\mathtt{A}_2^{\mathbf{no}}$) accepts exactly those trees T with $Q_1'(T) = \mathbf{yes}$ (resp. $Q_2'(T) = \mathbf{no}$).
(3) Construct the product automaton $\mathtt{B}$ of $\mathtt{A}_1^{\mathbf{yes}}$ and $\mathtt{A}_2^{\mathbf{no}}$, such that $\mathtt{B}$ accepts exactly those trees that are accepted by $\mathtt{A}_1^{\mathbf{yes}}$ and by $\mathtt{A}_2^{\mathbf{no}}$. Afterwards, check if the tree language recognised by $\mathtt{B}$ is empty. Note that this is the case if, and only if, $Q_1 \subseteq Q_2$.

Using time polynomial in the size of Q_1 and Q_2, Step (1) can be achieved in a standard way by appropriately extending the labelling alphabet Σ.

For Step (3), if $\mathtt{A}_1^{\mathbf{yes}}$ and $\mathtt{A}_2^{\mathbf{no}}$ are nondeterministic bottom-up tree automata, the construction of $\mathtt{B}$ takes time polynomial in the sizes of $\mathtt{A}_1^{\mathbf{yes}}$ and $\mathtt{A}_2^{\mathbf{no}}$, and the emptiness test can be done in time polynomial in the size of $\mathtt{B}$ (see e.g. [4]).

The first idea for tackling Step (2) is to use a standard translation of Boolean monadic datalog queries into monadic second-order (MSO) sentences: It is not difficult to see (cf., e.g. [8]) that any Boolean mDatalog(τ)-query Q can be translated in polynomial time into an equivalent MSO-sentence φ_Q of the form

$$\forall X_1 \cdots \forall X_n \ \exists z_1 \cdots \exists z_\ell \ \bigvee_{j=1}^{m} \gamma_j$$

where n is the number of intensional predicates of Q's monadic datalog program $\mathcal{P}$, ℓ and m are linear in the size of Q, and each γ_j is a conjunction of at most b atoms or negated atoms, where b is linear in the maximum number of atoms occurring in the body of a rule of $\mathcal{P}$. Applying the standard method for translating MSO-sentences into tree automata (cf., e.g., [11]), we can translate the sentence $\neg\varphi_Q$ into a nondeterministic bottom-up tree-automaton $\mathtt{A}^{\mathbf{no}}$ that accepts a tree T iff $Q(T) = \mathbf{no}$. This automaton has $2^{(m' \cdot c^{b'})}$ states, where m' and b' are linear in m and b, resp., and c is a constant not depending on Q or Σ; and $\mathtt{A}^{\mathbf{no}}$ can be constructed in time polynomial in $|\Sigma| \cdot 2^{n+\ell+m' \cdot c^{b'}}$.

Using the subset construction, one obtains an automaton $\mathtt{A}^{\mathbf{yes}}$ which accepts a tree T iff $Q(T) = \mathbf{yes}$; and this automaton has $2^{2^{(m' \cdot c^{b'})}}$ states.

Note that, a priori, b' might be linearly related to the size of Q. Thus, the approach described so far leads to a *3-fold exponential* algorithm that solves the QCP for unary mDatalog(τ)-queries.

In case that τ does *not* contain the **desc**-predicate, we obtain a 2-fold exponential algorithm as follows: At the end of Step (1) we rewrite Q_1' and Q_2' into queries that do not contain the **child**-predicate , and we transform both queries into *tree marking normal form* (TMNF), i.e., a normal form in which bodies of rules consist of at most two atoms, at least one of which is unary. From [8] we obtain that these transformations can be done in time polynomial in the size of Q_1' and Q_2'. Note that for TMNF-queries, the parameters b and b' are constant (i.e., they do not depend on the query), and thus the above description shows that for TMNF-queries the automaton $\mathtt{A}_2^{\mathbf{no}}$ can be constructed in 1-fold exponential time, and $\mathtt{A}_1^{\mathbf{yes}}$ can be constructed in 2-fold exponential time.

Finally, the key idea to obtain a *1-fold* exponential algorithm solving the QCP is to use a different construction for the automaton $\mathtt{A}_1^{\mathbf{yes}}$, which does not use the detour via an MSO-formula but, instead, takes a detour via a *two-way alternating tree automaton* (2ATA): We show that a Boolean TMNF-query can be translated, in polynomial time, into a 2ATA $\hat{\mathtt{A}}_1^{\mathbf{yes}}$ that accepts a tree T iff $Q_1(T) = \mathbf{yes}$. It is known that, within 1-fold exponential time, a 2ATA can be transformed into an equivalent nondeterministic bottom-up tree automaton (this was claimed already in [5]; detailed proofs of more general results can be found in [12,10]). In summary, this leads to a 1-fold exponential algorithm for solving the QCP for mDatalog($\tau_{GK}^{\mathbf{child}}$) on ordered trees. $\square$

Since $\tau_u^{\mathbf{root},\mathbf{leaf}} \subseteq \tau_{GK}^{\mathbf{child}}$, Theorem 3 immediately implies:

Corollary 4 *The QCP for unary* mDatalog($\tau_u^{\mathbf{root},\mathbf{leaf}}$) *on unordered trees belongs to* EXPTIME.

It remains open if the EXPTIME-membership results of Theorem 3 and Corollary 4 can be generalised to queries that also use the descendant predicate **desc**. However, the first approach described in the proof of Theorem 3 yields a 3-fold exponential algorithm. We can improve this by using methods and results from [8] and [9] to eliminate the **desc**-predicate at the expense of an exponential blow-up of the query size. Afterwards, we apply the algorithms provided by Theorem 3 and Corollary 4. This leads to the following:

Theorem 5 *The QCP for unary* mDatalog($\tau_u^{\mathbf{root},\mathbf{leaf},\mathbf{desc}}$) *on unordered trees and for unary* mDatalog($\tau_{GK}^{\mathbf{child},\mathbf{desc}}$) *on ordered trees can be solved in 2-fold exponential time.*

Open Question. It remains open to close the gap between the EXPTIME lower and the 2EXPTIME upper bound for the case where the *descendant*-axis is involved.

Acknowledgment. The first author would like to thank Mariano Zelke for countless inspiring discussions and helpful hints on and off the topic.

References

1. S. Abiteboul, P. Bourhis, A. Muscholl, and Z. Wu. Recursive queries on trees and data trees. In *Proc. ICDT'13*, pages 93–104, 2013.
2. M. Benedikt, P. Bourhis, and P. Senellart. Monadic datalog containment. In *Proc. ICALP'12*, pages 79–91, 2012.
3. B. S. Chlebus. Domino-tiling games. *J. Comput. Syst. Sci.*, 32(3):374–392, 1986.
4. H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available at `http://www.grappa.univ-lille3.fr/tata`, 2008. release November, 18th 2008.
5. S. Cosmadakis, H. Gaifman, P. Kanellakis, and M. Vardi. Decidable optimization problems for database logic programs. In *Proc. STOC'88*, pages 477–490, 1988.
6. E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and expressive power of logic programming. *ACM Comput. Surv.*, 33(3):374–425, 2001.
7. A. Frochaux and N. Schweikardt. A note on monadic datalog on unranked trees. *Technical Report, available at CoRR*, abs/1310.1316, 2013.
8. G. Gottlob and C. Koch. Monadic datalog and the expressive power of languages for web information extraction. *J. ACM*, 51(1):74–113, 2004.
9. G. Gottlob, C. Koch, and K. Schulz. Conjunctive queries over trees. *J. ACM*, 53(2):238–272, 2006.
10. S. Maneth, S. Friese, and H. Seidl. Type-Checking Tree Walking Transducers. In D. D'Souza and P. Shankar, editors, *Modern applications of automata theory*, volume 2 of *IISc Research Monographs*. World Scientific, 2010.
11. W. Thomas. Languages, automata, and logic. In *Handbook of Formal Languages*, volume 3, pages 389–455. Springer-Verlag, 1997.
12. M. Vardi. Reasoning about the past with two-way automata. In *Proc. ICALP'98*, pages 628–641, 1998.