Monadic Datalog Containment on Trees Using the Descendant-Axis

André Frochaux and Nicole Schweikardt

Institut für Informatik, Humboldt-Universität zu Berlin
`{frochaua,schweikn}@informatik.hu-berlin.de`

Abstract. In their AMW'14-paper, Frochaux, Grohe, and Schweikardt showed that the query containment problem for monadic datalog on finite unranked labeled trees is Exptime-complete when (a) considering unordered trees using the *child*-axis, and when (b) considering ordered trees using the axes *firstchild*, *nextsibling*, and *child*. Furthermore, when allowing to use also the *descendant*-axis, the query containment problem was shown to be solvable in 2-fold exponential time, but it remained open to determine the problem's exact complexity in presence of the descendant-axis. The present paper closes this gap by showing that, in the presence of the descendant-axis, the problem is 2Exptime-hard.

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 [10] showed that on *ordered* trees the QCP for monadic datalog is Exptime-hard and decidable, leaving open the question for a tight bound. This gap was closed by Frochaux, Grohe, and Schweikardt in [8] by giving a matching Exptime upper bound for the QCP for monadic datalog on ordered trees using the axes *firstchild*, *nextsibling*, and *child*. Similar results were obtained in [8] also for *unordered* finite labeled trees: in this setting, the QCP is Exptime-complete for monadic datalog queries on unordered trees using the *child*-axis.

For the case where queries are allowed to also use the *descendant*-axis, [8] presented a 2-fold exponential time algorithm for the QCP for monadic datalog on (ordered or unordered) trees. Determining the problem's exact complexity in the presence of the *descendant*-axis, however, was left open.

The present paper closes the gap by proving a matching 2Exptime lower bound (both, for ordered and for unordered trees). This gives a conclusive answer to a question posed by Abiteboul et al. in [1], asking for the complexity of the QCP on unordered trees in the presence of the descendant-axis. Our 2Exptime-hardness proof for *ordered* trees is by a reduction from a 2Exptime-hardness result of [3] for the validity of conjunctive queries w.r.t. schema constraints. For obtaining the 2EXPTIME-hardness on *unordered* trees, we follow the approach of [3] and construct a reduction from the 2EXPTIME-complete word problem for exponential-space bounded alternating Turing machines [4].

The remainder of the paper is organised as follows. Section 2 fixes the basic notation. Section 3 presents a 2EXPTIME lower bound for the QCP on ordered trees using the axes *firstchild*, *nextsibling*, *root*, *leaf*, *lastsibling*, *child*, *descendant*. Section 4 is devoted to the 2EXPTIME lower bound for the QCP on unordered trees using only the axes *child* and *descendant*. We conclude in Section 5.

Due to space limitations, many proof details had to be deferred to the paper's full version.

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 τ-*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 $R(a_1, \dots, a_{ar(R)})$ for all relation symbols $R \in \tau$ and all tuples $(a_1, \dots, a_{ar(R)}) \in R^{\mathcal{A}}$.

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 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 non-empty 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 directed 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 $\tau_u := \{\mathbf{child}\}$.

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 $\mathbf{label}_\alpha$, $\mathbf{fc}$, and $\mathbf{ns}$. We fix the schemas $\tau_o := \{\mathbf{fc}, \mathbf{ns}\}$ and $\tau_{GK} := \tau_o^{\mathbf{root},\mathbf{leaf},\mathbf{ls}}$. In [10], Gottlob and Koch used $\tau_{GK,\Sigma}$-structures to represent ordered Σ-labeled trees.

Datalog. We assume that the reader is familiar with the syntax and semantics of *datalog* (cf., e.g., [6, 10]). 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 $\text{idb}(\mathcal{P})$ and $\text{edb}(\mathcal{P})$ we denote the sets of intensional and extensional predicates of $\mathcal{P}$, resp. We say that $\mathcal{P}$ *is of schema* τ if $\text{edb}(\mathcal{P}) \subseteq \tau$. 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 $\mathcal{P}$. 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 $Q_{Bool}(T) = \mathbf{yes}$ 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. 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 [10] — for short, we will say "$\text{mDatalog}(\tau_{GK}) = \text{MSO}(\tau_{GK})$ on ordered trees". Since the **child** and **desc** relations are definable in $\text{MSO}(\tau_{GK})$, $\text{mDatalog}(\tau_{GK}) = \text{mDatalog}(\tau_{GK}^{\mathbf{child},\mathbf{desc}})$ on ordered trees. Moreover, for (ordered or unordered) trees, every monadic Datalog query that uses the **desc**-axis can be rewritten in 1-fold exponential time into an equivalent monadic datalog query which uses the **child**-axis, but not the **desc**-axis (see the proof of Lemma 23 in the full version of [8]).

Using the monotonicity of the immediate consequence operator, one obtains that removing any of the predicates $\mathbf{root}, \mathbf{leaf}, \mathbf{ls}$ from τ_{GK} strictly decreases the expressive power of mDatalog on ordered trees (see [9]). By a similar reasoning one also obtains that on *unordered* trees, represented as $\tau_{u,\Sigma}^{\mathbf{root},\mathbf{leaf},\mathbf{desc}}$-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 [9].

The Query Containment Problem (QCP). Let τ_Σ be one of the schemas used 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 $Q_1(T) = \mathbf{yes}$ then also $Q_2(T) = \mathbf{yes}$. We write $Q_1 \not\subseteq Q_2$ to indicate that $Q_1 \subseteq Q_2$ does not hold. For a schema τ, the *query containment problem (QCP) for* mDatalog(τ) *on finite labeled trees* receives as input a finite alphabet Σ and two (unary or Boolean) mDatalog(τ_Σ)-queries Q_1 and Q_2, and the task is to decide whether $Q_1 \subseteq Q_2$. From [8] we know:

Theorem 1 (Frochaux et al. [8]) *The QCP for* mDatalog($\tau_u^{\mathbf{root},\mathbf{leaf},\mathbf{desc}}$) *on unordered trees and for* mDatalog($\tau_{GK}^{\mathbf{child},\mathbf{desc}}$) *on ordered trees can be solved in 2-fold exponential time.*

3 2Exptime-hardness on Ordered Trees

Theorem 2 *The QCP for Boolean* mDatalog($\tau_{GK}^{\mathbf{child},\mathbf{desc}}$) *on finite labeled ordered trees is* 2Exptime*-hard.*

The proof is by a reduction based on a 2Exptime-hardness result of Björklund, Martens, and Schwentick [3]. For stating their result, we recall some notation used in [3]. A *nondeterministic (unranked) tree automaton* (NTA) $\mathtt{A} = (\Sigma, S, \Delta, F)$ consists of an input alphabet Σ, a finite set S of states, a set $F \subseteq S$ of accepting states, and a finite set Δ of transition rules of the form $(s, \alpha) \to L$, where $s \in S$, $\alpha \in \Sigma$, and L is a regular string-language over S. A *run* of the NTA $\mathtt{A}$ on a ordered Σ-labeled tree T is a mapping $\rho : V^T \to S$ such that the following is true for all nodes v of T, where α denotes the label of v in T: if v has $n \geqslant 0$ children $u_1, \dots, u_n$ (in order from the left to the right), then there exists a rule $(s, \alpha) \to L$ in Δ such that $\rho(v) = s$ and $w_v \in L$, for the string $w_v := \rho(u_1) \cdots \rho(u_n)$. In particular, if v is a leaf, then there must be a rule $(s, \alpha) \to L$ in Δ such that $\rho(v) = s$ and $\varepsilon \in L$, where ε denotes the empty string.

A run ρ of $\mathtt{A}$ on T is *accepting*, if T's root note v is labeled with an accepting state of $\mathtt{A}$, i.e., $\rho(v) \in F$. A finite ordered Σ-labeled tree T is *accepted* by $\mathtt{A}$, if there exists an accepting run of $\mathtt{A}$ on T. We write $L(\mathtt{A})$ to denote the *language* of $\mathtt{A}$, i.e., the set of all finite ordered Σ-labeled trees that are accepted by $\mathtt{A}$.

To present an NTA $\mathtt{A} = (\Sigma, S, \Delta, F)$ as an input for an algorithm, the string-languages L that occur in the right-hand side of rules in Δ are specified by NFAs $\mathtt{A}_L = (\Sigma_L, Q_L, \delta_L, q_L, F_L)$, whose input alphabet is $\Sigma_L := S$, and where Q_L is a finite set of states, $\delta_L \subseteq (Q_L \times \Sigma_L \times Q_L)$ is a transition relation, $q_L \in Q_L$ is the initial state, and $F_L \subseteq Q_L$ is the set of accepting states of $\mathtt{A}_L$. The *size of* $\mathtt{A}_L$ is

$\|\mathtt{A}_L\| := |Q_L| + |\delta_L|$, and the *size of* $\mathtt{A}$ is the sum of $|\Sigma|$, $|S|$, $|\Delta|$, and $\|\mathtt{A}_L\|$, for all $L \in \mathrm{strL}(\mathtt{A})$, where $\mathrm{strL}(\mathtt{A})$ is the set of all string-languages L that occur in the right-hand side of a rule in Δ.

In [3], NTAs are used to describe schema information. A Boolean query Q is said to be *valid with respect to* an NTA $\mathtt{A}$ if $Q(T) = \mathbf{yes}$ for every ordered Σ-labeled tree $T \in L(\mathtt{A})$. The particular queries of interest here are *Boolean CQ*$(\mathbf{child}, \mathbf{desc})$ queries, i.e., Boolean conjunctive queries of schema $\tau^{\mathbf{desc}}_{u,\Sigma} = \{\mathbf{child}, \mathbf{desc}\} \cup \{\mathbf{label}_\alpha : \alpha \in \Sigma\}$, for a suitable alphabet Σ. The problem *"validity of Boolean CQ*$(\mathbf{child}, \mathbf{desc})$ *w.r.t. a tree automaton"* receives as input a Boolean CQ$(\mathbf{child}, \mathbf{desc})$ query Q and an NTA $\mathtt{A}$, and the task is to decide whether Q is valid with respect to $\mathtt{A}$.

Theorem 3 (Björklund et al. [3]) *Validity of Boolean CQ*$(\mathbf{child}, \mathbf{desc})$ *w.r.t. a tree automaton is* 2EXPTIME*-complete.*

Our proof of Theorem 2 is via a polynomial-time reduction from the problem *validity of Boolean CQ*$(\mathbf{child}, \mathbf{desc})$ *w.r.t. a tree automaton* to the *QCP for Boolean* mDatalog$(\tau^{\mathbf{child},\mathbf{desc}}_{GK})$ *on finite labeled ordered trees.*

Let Q_{CQ} be a Boolean CQ$(\mathbf{child}, \mathbf{desc})$-query, and let $\mathtt{A}$ be an NTA with input alphabet Σ. We translate Q_{CQ} into an equivalent mDatalog$(\tau^{\mathbf{desc}}_{u,\Sigma})$-query $Q'_{\mathrm{CQ}} = (\mathcal{P}, P)$: If Q_{CQ} is of the form $\mathit{Ans}() \leftarrow R_1(u_1), \ldots, R_\ell(u_\ell)$ for relational atoms $R_1(u_1), \ldots, R_\ell(u_\ell)$, we choose an arbitrary variable x that occurs in at least one of these atoms, we use a new unary idb-predicate P, and we let $\mathcal{P}$ be the program consisting of the two rules $P(x) \leftarrow R_1(u_1), \ldots, R_\ell(u_\ell)$ and $P(x) \leftarrow \mathbf{child}(x, y), P(y)$.

Then, for every ordered Σ-labeled tree T we have $Q'_{\mathrm{CQ},\mathit{Bool}}(T) = \mathbf{yes}$ iff $Q_{\mathrm{CQ}}(T) = \mathbf{yes}$. The following Lemma 4 constructs, in time polynomial in the size of $\mathtt{A}$, an mDatalog$(\tau^{\mathbf{child}}_{GK,\Sigma})$-query $Q_{\mathtt{A}}$ which is equivalent to $\mathtt{A}$, i.e., for every ordered Σ-labeled tree T we have $Q_{\mathtt{A},\mathit{Bool}}(T) = \mathbf{yes}$ iff $T \in L(\mathtt{A})$.

Note that Q_{CQ} is valid w.r.t. $\mathtt{A}$ if, and only if, $Q_{\mathtt{A},\mathit{Bool}} \subseteq Q'_{\mathrm{CQ},\mathit{Bool}}$. Thus, we obtain the desired polynomial-time reduction, showing that the QCP for Boolean mDatalog$(\tau^{\mathbf{child},\mathbf{desc}}_{GK})$ on finite ordered Σ-labeled trees inherits the 2EXPTIME-hardness from the problem "validity of Boolean CQ$(\mathbf{child}, \mathbf{desc})$ w.r.t. a tree automaton". All that remains to finish the proof of Theorem 2 is to prove the following Lemma 4.

Lemma 4 *For every NTA* $\mathtt{A} = (\Sigma, S, \Delta, F)$ *there is an* mDatalog$(\tau^{\mathbf{child}}_{GK,\Sigma})$*-query* $Q = (\mathcal{P}, P)$*, such that for every finite ordered* Σ*-labeled tree* T *we have* $Q_{\mathit{Bool}}(T) = \mathbf{yes}$ *iff* $T \in L(\mathtt{A})$*. Furthermore,* Q *is constructible from* $\mathtt{A}$ *in time polynomial in the size of* $\mathtt{A}$*.*

Proof. We construct a monadic datalog program $\mathcal{P}$ which, for every node v of T, computes information on *all* states that $\mathtt{A}$ can assume at node v, i.e., all states $s \in S$ for which there is a run ρ of $\mathtt{A}$ on the subtree of T rooted at v, such that $\rho(v) = s$. To this end, for every state $s \in S$, we will use an idb-predicate s. The query Q_{Bool} will accept an input tree T if there is an accepting state $s \in F$

such that $s(\mathit{root}^T) \in \mathcal{T}_\mathcal{P}^\omega(T)$, where root^T denotes the root of T. The program $\mathcal{P}$ is constructed in such a way that it performs a generalised version of the well-known *powerset construction*.

Recall that the transition rules of $\mathtt{A}$ are of the form $(s,\alpha) \to L$, where $s \in S$, $\alpha \in \Sigma$, and L is a regular string-language over S, specified by an NFA $\mathtt{A}_L = (\Sigma_L, Q_L, \delta_L, q_L, F_L)$ with $\Sigma_L = S$ and $\delta_L \subseteq (Q_L \times \Sigma_L \times Q_L)$. W.l.o.g., we assume that the state sets of all the NFAs are mutually disjoint, and disjoint with S.

To emulate the standard powerset construction of the NFA $\mathtt{A}_L$, we use an idb-predicate q for every state $q \in Q_L$, and an extra idb-predicate Acc_L. If $u_1, \dots, u_n$ are the children of a node v in an input tree T, the NFA $\mathtt{A}_L$ processes the strings over alphabet S that are of the form $s_1 \cdots s_n$, where s_i is a state that $\mathtt{A}$ can assume at node u_i (for every $i \in \{1,\dots,n\}$). We start by letting $\mathcal{P}_L := \emptyset$ and then add to $\mathcal{P}_L$ the following rules: For the initial state q_L of $\mathtt{A}_L$, consider all $s \in S$ and $q \in Q_L$ such that $(q_L, s, q) \in \delta_L$, and add to $\mathcal{P}_L$ the rule

$$q(x) \ \leftarrow\ \mathbf{fc}(y,x),\ s(x)\,.$$

Afterwards, for every transition $(q,s,q') \in \delta_L$, add to $\mathcal{P}_L$ the rule

$$q'(x') \ \leftarrow\ q(x),\ \mathbf{ns}(x,x'),\ s(x')\,.$$

Finally, for every accepting state $q \in F_L$ of $\mathtt{A}_L$, add to $\mathcal{P}_L$ the rule

$$\mathit{Acc}_L(x) \ \leftarrow\ \mathbf{ls}(x),\ q(x)\,.$$

Clearly, the program $\mathcal{P}_L$ can be constructed in time polynomial in $\|\mathtt{A}_L\|$.

Now, we are ready to construct the monadic datalog program $\mathcal{P}$ that simulates the NTA $\mathtt{A}$. We start by letting $\mathcal{P}$ be the disjoint union of the programs $\mathcal{P}_L$, for all $L \in \mathrm{strL}(\mathtt{A})$. The computation of $\mathtt{A}$ on an input tree T starts in the leaves of T. Thus, to initiate the simulation of $\mathtt{A}$, we consider every rule $(s,\alpha) \to L$ in Δ, where $\varepsilon \in L$.[1] For each such rule, we add to $\mathcal{P}$ the rule

$$s(x) \ \leftarrow\ \mathbf{label}_\alpha(x),\ \mathbf{leaf}(x)\,.$$

Note that for each $L \in \mathrm{strL}(\mathtt{A})$, the program $\mathcal{P}_L$ ensures that every *last* sibling u_n of a node v will be marked by $\mathit{Acc}_L(u_n)$ iff the states of $\mathtt{A}$ assigned to u_n and its siblings form a string in L. To transfer this information from the last sibling to its parent node, we add to $\mathcal{P}$ the rule

$$\mathbf{child}_{\mathit{Acc}_L}(y) \ \leftarrow\ \mathbf{child}(y,x),\ \mathbf{ls}(x),\ \mathit{Acc}_L(x)\,,$$

where $\mathbf{child}_{\mathit{Acc}_L}$ is a new idb-predicate, for every $L \in \mathrm{strL}(\mathtt{A})$.
Afterwards, we consider every rule $(s,\alpha) \to L$ in Δ, and add to $\mathcal{P}$ the rule

$$s(x) \ \leftarrow\ \mathbf{child}_{\mathit{Acc}_L}(x),\ \mathbf{label}_\alpha(x)\,.$$

Finally, to test if $\mathtt{A}$ accepts an input tree T, we add rules to test whether T's root is assigned an accepting state of $\mathtt{A}$. To this end, we consider every accepting state $s \in F$ of $\mathtt{A}$ and add to $\mathcal{P}$ the rule

$$P(x) \ \leftarrow\ \mathbf{root}(x),\ s(x)\,.$$

This finishes the construction of the program $\mathcal{P}$ and the query $Q = (\mathcal{P}, P)$. Clearly, $\mathcal{P}$ is a monadic datalog program of schema $\tau_{GK,\Sigma}^{\mathbf{child}}$, and Q can be constructed in time polynomial in $\|A\|$. It is not difficult, but somewhat tedious, to

[1] Note that "$\varepsilon \in L\,$?" can be checked by simply checking whether $q_L \in F_L$.

verify that, as intended by the construction, indeed for every finite ordered Σ-labeled tree T we have $Q_{Bool}(T) = \mathbf{yes}$ if, and only if, there exists an accepting run of the NTA $\mathtt{A}$ on T. This completes the proof of Lemma 4. $\square$

4 2EXPTIME-hardness on Unordered Trees

Our next aim is to transfer the statement of Theorem 2 to *unordered* trees. Precisely, we will show the following.

Theorem 5 *The QCP for Boolean* mDatalog($\tau_u^{\mathbf{desc}}$) *on finite labeled unordered trees is* 2EXPTIME*-hard.*

For proving Theorem 5, we cannot directly build on Björklund et al.'s Theorem 3, since their NTAs explicitly refer to *ordered* trees.

By constructing suitable reductions, we can show that proving Theorem 5 boils down to proving the following Theorem 6, which deals with the *emptiness problem* on trees over a *ranked* alphabet.

For the remainder of this section, Σ' will denote a *ranked* finite alphabet. I.e., Σ' is a finite set of symbols, and each symbol $\alpha \in \Sigma'$ is equipped with a fixed *arity* $ar(\alpha) \in \mathbb{N}$. An unordered *ranked* Σ'-labeled tree is an unordered Σ'-labeled tree where each node labeled with symbol $\alpha \in \Sigma'$ has exactly $ar(\alpha)$ children. For a Boolean mDatalog($\tau_{u,\Sigma'}^{\mathbf{desc}}$)-query Q, we say that Q is *unsatisfiable by unordered ranked trees* (in symbols: $Q = \varnothing$) if for every finite unordered ranked Σ'-labeled tree T we have $Q(T) = \emptyset$. The *emptiness problem for Boolean* mDatalog($\tau_{u,\Sigma'}^{\mathbf{desc}}$) *on finite unordered ranked Σ'-labeled trees* receives as input a Boolean mDatalog($\tau_{u,\Sigma'}^{\mathbf{desc}}$)-query Q, and the task is to decide whether $Q = \varnothing$. The main technical step needed for proving Theorem 5 is to prove the following.

Theorem 6 *There is a ranked finite alphabet Σ', such that the emptiness problem for Boolean* mDatalog($\tau_{u,\Sigma'}^{\mathbf{desc}}$) *on finite unordered ranked Σ'-labeled trees is* 2EXPTIME*-hard.*

For the proof of Theorem 6, we can build on the approach used by Björklund et al. for proving Theorem 3: As in [3], we proceed by a reduction from the word problem for exponential-space bounded alternating Turing machines, which is known to be 2EXPTIME-complete [4]. The remainder of this section is devoted to the proof of Theorem 6.

An *alternating Turing machine* (ATM) is a nondeterministic Turing machine $\mathtt{A} = (Q, \Sigma, \Gamma, \delta, q_0)$ whose state space Q is partitioned into *universal states* $Q_\forall$, *existential states* $Q_\exists$, an accepting state q_a, and a *rejecting state* q_r. The ATM's tape cells are numbered 0,1,2,.... A *configuration* of $\mathtt{A}$ is a finite string of the form $w_1 q w_2$ with $w_1, w_2 \in \Gamma^*$ and $q \in Q$, representing the situation where the ATM's tape contains the word $w_1 w_2$, followed by blanks, the ATM's current state is q, and the head is positioned at the first letter of w_2. A configuration $w_1 q w_2$ is a *halting* (*universal*, *existential*, resp.) configuration if $q \in \{q_a, q_r\}$ ($q \in Q_\forall$, $q \in Q_\exists$, resp.). W.l.o.g., no halting configuration has a successor configuration, and every halting configuration is of the form qw. A *computation tree* $T_{\mathtt{A}}$ of the ATM $\mathtt{A}$ on input $w \in \Sigma^*$ is a tree labeled with configurations of $\mathtt{A}$, such that the root of $T_{\mathtt{A}}$ is labeled by $q_0 w$, and for each node v of $T_{\mathtt{A}}$ labeled by $w_1 q w_2$,

- if $q \in Q_\exists$, then u has exactly one child, and this child is labeled with a successor configuration of $w_1 q w_2$,
- if $q \in Q_\forall$, then u has a child v for every successor configuration $w_1' q' w_2'$, and v is labeled by $w_1' q' w_2'$,
- if $q \in \{q_a, q_r\}$, then u is a leaf of $T_{\mathtt{A}}$.

A computation tree is *accepting* if all its branches are finite and all its leaves are labeled by configurations with state q_a. The *language* $L(\mathtt{A})$ of $\mathtt{A}$ is defined as the set of all words $w \in \Sigma^*$, for which there exists an accepting computation tree of $\mathtt{A}$ on w. W.l.o.g., we will assume that the ATM is *normalized*, i.e., every non-halting configuration has precisely two successor configurations, each universal step only affects the state of the machine, and the machine always alternates between universal and existential states.

The proof of Theorem 6 proceeds by a reduction from the word problem for exponential-space bounded ATMs $\mathtt{A}$. The reduction itself will be done from an ATM with empty input word. To this end, we construct, in the canonical way, for the given exponential-space bounded ATM $\mathtt{A}$ and the given word $w \in \Sigma^*$ an ATM $\mathtt{A}_w$ that works in space exponential in the size of w and accepts the empty word if, and only if, $\mathtt{A}$ accepts w. Since $\mathtt{A}$ is exponential-space bounded, the non-blank portion of the ATM's tape during a computation of $\mathtt{A}_w$ will never be longer that 2^n, where n is polynomial in the size $|w|$ of the original input.

The crucial point of the reduction is to find an encoding of computation trees of $\mathtt{A}_w$ on empty input, which can be verified by a mDatalog($\tau^{\mathbf{desc}}_{u,\Sigma'}$)-query that can be constructed in time *polynomial* in the size of $\mathtt{A}_w$. For this, it is necessary to find a smart encoding of the tape inscription of length 2^n. This encoding shall allow to compare the content of every tape cell with the same tape cell of the successor configuration. To achieve this, we adapt the encoding of Björklund et al. [3]; in particular, we use their very elegant "navigation gadgets".

We choose a fixed ranked finite alphabet Σ' which, among other symbols, contains a 0-ary symbol $\bot$, unary symbols $r, p, m, 0, 1$, binary symbols $\mathrm{CT}_\exists^{left}, \mathrm{CT}_\exists^{right}$, and 3-ary symbols $\mathrm{CT}_\forall$ and s. Consider a computation tree $T_{\mathtt{A}_w}$ of a normalized ATM $\mathtt{A}_w = (Q, \Sigma, \Gamma, \delta, q_0)$, see Figure 1.

We fix an arbitrary order on the children of nodes in $T_{\mathtt{A}_w}$, such that every universal node has a left child and a right child. The *encoding* $T := \mathrm{enc}(T_{\mathtt{A}_w})$ is the ranked Σ'-labeled unordered tree obtained from $T_{\mathtt{A}_w}$ by replacing every node v labeled $w_1 q w_2$ with a Σ'-labeled ranked tree $\mathrm{enc}(t_v)$, as follows:

- if v is universal, then the root of $\mathrm{enc}(t_v)$ is labeled with $\mathrm{CT}_\forall$,
- if v is existential, and v is the root of $T_{\mathtt{A}_w}$ or v is the left child of a universal node, then the root of $\mathrm{enc}(t_v)$ is labeled with $\mathrm{CT}_\exists^{left}$,
- if v is existential, and v is the right child of a universal node, then the root of $\mathrm{enc}(t_v)$ is labeled with $\mathrm{CT}_\exists^{right}$,

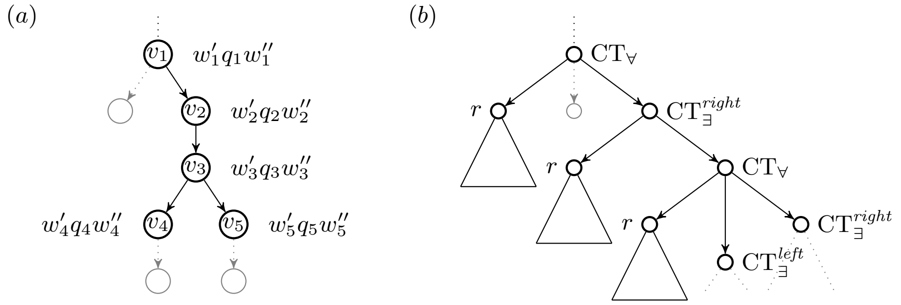

Fig. 1. (a) A part of a computation tree $T_{\mathtt{A}_w}$, where the node v_1 labeled by $w'_1q_1w''_1$ is universal, and its children are existential. The node v_2 labeled by $w'_2q_2w''_2$ is the right child of v_1. The node v_2 has one child, the univeral node v_3. (b) The replacement of v_1 is a tree with a root node labeled by $\mathrm{CT}_\forall$ and with three children, the first is labeled by r and is the root of the subtree encoding the configuration in v_1, the second is the replacement for its left child, and the third is the replacment for its right child. The obtained tree $T := \mathrm{enc}(T_{\mathtt{A}_w})$ is an unordered ranked Σ'-labeled tree.

– exactly one child of the root of $\mathrm{enc}(t_v)$ is labeled by r (this will be the root of the subtree that encodes the configuration at v), and
– for each child u of v in $T_{\mathtt{A}_w}$, $\mathrm{enc}(t_v)$ has a subtree $\mathrm{enc}(t_u)$, which is the encoded subtree of $T_{\mathtt{A}_w}$ obtained by the replacement of u.

The subtree γ_r rooted at the r-labeled child of the root of $\mathrm{enc}(t_v)$, encodes the configuration $c := w_1qw_2$ represented by node v in $T_{\mathtt{A}_w}$. Since $\mathtt{A}$ is exponential-space bounded, the tape inscription of c has length $\leqslant 2^n$. For representing c, we use a full binary ordered tree of height n. The path from the root to a leaf specifies the address of the tape cell represented by the leaf, and the leaf carries information on the tape cell's inscription and, in case that the tape cell is the current head position, also information on the current state; all this information is encoded by a suitable *tape cell gadget* that is attached to the "leaf". The number k of possible tape cell inscriptions (enriched with information on the current state) is *polynomial* in $\|\mathtt{A}_w\|$. The nodes of the "full binary tree" are called *skeleton nodes* and are labeled s. To ensure that the desired query Q can be constructed in *polynomial* time, we attach to each skeleton node a *navigation gadget* [3], which is a path of length 4. To indicate that a node is a left (resp., right) child, this gadget is labeled $p-0-1-\bot$ (resp., $p-1-0-\bot$). See Figure 2 for an illustration of the *navigation gadget* and the *tape cell gadget*.

Given an ATM $\mathtt{A}$ and a word $w \in \Sigma^*$, we construct in polynomial time an mDatalog($\tau^{\mathbf{desc}}_{u,\Sigma'}$)-query $Q = (\mathcal{P}, Ans)$ such that $Q_{Bool} \neq \varnothing$ iff there is an accepting computation tree for $\mathtt{A}_w$ on ε, i.e., $w \in L(\mathtt{A})$. The query Q consists of two parts, one to verify that the structure of the input tree represents an encoded computation tree, and the other to verify consistency with the ATM's transition relation. Details can be found in the paper's full version. The particular choice of the navigation gadgets ensures that Q can be constructed in time *polynomial*

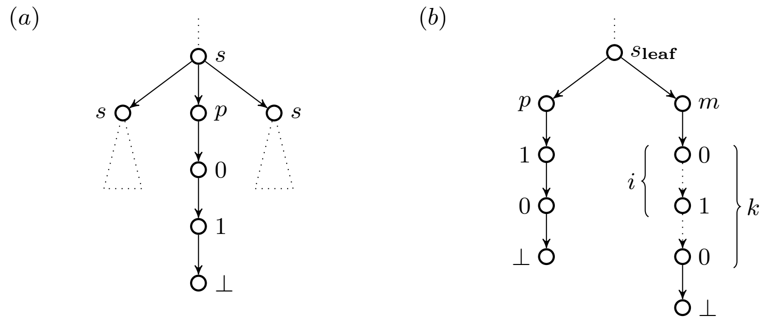

Fig. 2. (a) A skeleton node and its navigation gadget, indicating that the node is its parent's left child. (b) A skeleton node encoding a leaf of the configuration tree. This leaf is its parent's right child. It has a tape cell gadget m followed by k digits, the i-th of which is labeled with 1 iff the tape cell's inscription is represented by the number i.

in the size of `A` and w. The only point where we make essential use of the **desc**-predicate is during the comparison of the cells by using the navigation gadgets.

5 Final Remarks

Along with the upper bound provided by Theorem 1, and since $\tau_u^{\mathbf{desc}} \subseteq \tau_o^{\mathbf{child},\mathbf{desc}}$, Theorem 5 implies the following corollary, which summarizes our main results.

Corollary 7 *The QCP is* 2EXPTIME*-complete for Boolean* mDatalog($\tau_u^{\mathbf{desc}}$) *on finite labeled unordered trees, and for Boolean* mDatalog($\tau_o^{\mathbf{child},\mathbf{desc}}$) *on finite labeled ordered trees.*

By applying standard reductions, the 2EXPTIME-completeness results of Corollary 7 carry over from the QCP to the *equivalence problem*. When restricting attention to *ranked* trees over a ranked finite alphabet, the 2EXPTIME-completeness results also carry over to the *emptiness problem*. For *unranked* labeled trees, the emptiness problem for mDatalog($\tau_o^{\mathbf{child},\mathbf{desc}}$) is in 2EXPTIME, but we currently do not have a matching 2EXPTIME-hardness result.

An overview of the currently known results is given in Table 1; for further information and detailed proofs we refer to [7].

Table 1. Complexity of monadic datalog on finite labeled trees; $N \subseteq \{\mathbf{root}, \mathbf{leaf}\}$ and $M \subseteq \{\mathbf{root}, \mathbf{leaf}, \mathbf{ls}, \mathbf{child}\}$; "c" ("h") means "complete" ("hard").

	τ_u^N	τ_o^M	$\tau_u^{N\cup\{\mathbf{desc}\}}$	$\tau_o^{M\cup\{\mathbf{child},\mathbf{desc}\}}$	$\tau_{GK}^{\mathbf{child},\mathbf{desc}}$	
Emptiness	EXPTIME-c		EXPTIME-h & in 2EXPTIME			*unranked*
			2EXPTIME-c			*ranked*
Equivalence	EXPTIME-c		2EXPTIME-c			*unranked*
						ranked
Containment	EXPTIME-c		2EXPTIME-c			*unranked*
						ranked

References

[1] Abiteboul, S., Bourhis, P., Muscholl, A., Wu, Z.: Recursive queries on trees and data trees. In: Proc. ICDT'13. pp. 93–104 (2013)

[2] Benedikt, M., Bourhis, P., Senellart, P.: Monadic datalog containment. In: Proc. ICALP'12. pp. 79–91 (2012)

[3] Björklund, H., Martens, W., Schwentick, T.: Optimizing conjunctive queries over trees using schema information. In: Proc. MFCS'08. pp. 132–143 (2008), full version: `http://www8.cs.umu.se/~henrikb/papers/mfcs08full.pdf` (accessed: 2016-03-05)

[4] Chandra, A.K., Kozen, D., Stockmeyer, L.J.: Alternation. J. ACM 28(1), 114–133 (1981), `http://doi.acm.org/10.1145/322234.322243`

[5] Cosmadakis, S., Gaifman, H., Kanellakis, P., Vardi, M.: Decidable optimization problems for database logic programs. In: Proc. STOC'88. pp. 477–490 (1988)

[6] Dantsin, E., Eiter, T., Gottlob, G., Voronkov, A.: Complexity and expressive power of logic programming. ACM Comput. Surv. 33(3), 374–425 (2001)

[7] Frochaux, A.: Static Analysis of Monadic Datalog on Finite Labeled Trees. Doctoral Dissertation, Humboldt-Universität zu Berlin, to appear

[8] Frochaux, A., Grohe, M., Schweikardt, N.: Monadic datalog containment on trees. In: Proc. AMW'14 (2014), full version: `http://arxiv.org/abs/1404.0606`

[9] Frochaux, A., Schweikardt, N.: A note on monadic datalog on unranked trees. Technical Report, available at `http://arxiv.org/abs/1310.1316` (2013)

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