A note on monadic datalog on unranked trees

André Frochaux Nicole Schweikardt

Goethe-Universität Frankfurt am Main
{afrochaux|schweika}@informatik.uni-frankfurt.de

Version: October 7, 2013

Abstract

In the article *Recursive queries on trees and data trees* (ICDT'13), Abiteboul *et al.* asked whether the containment problem for monadic datalog over unordered unranked labeled trees using the child relation and the descendant relation is decidable. This note gives a positive answer to this question, as well as an overview of the relative expressive power of monadic datalog on various representations of unranked trees.

1 Introduction

The logic and database theory literature has considered various kinds of representations of finite labeled trees as logical structures. In particular, trees are either ranked or unranked (i.e., the number of children of each node is bounded by a constant, or unbounded); the children of each node are either ordered or unordered; and there is or there is not available the descendant relation (i.e., the transitive closure of the child relation); for overviews see [8, 9, 13, 4].

Considering *ordered* unranked labeled trees, Gottlob and Koch [5] showed that monadic datalog, viewed as a language for defining Boolean or unary queries on such trees, is exactly as expressive as monadic second-order logic. For achieving this result, they represent a tree as a logical structure where the nodes of the tree form the structure's universe, on which there are available the firstchild relation, the nextsibling relation, and unary relations for representing the root, the leaves, the last siblings, and the labels of the nodes. Other papers, e.g. [10, 3], consider representations of trees where also the child relation and its transitive closure, the descendant relation are available.

For *unordered* unranked labeled trees, one usually considers logical representations consisting only of the child relation, and possibly also the descendant relation, along with unary relations for encoding the node labels, cf. e.g. [1, 6, 2]. Recently, Abiteboul *et al.* [1] considered recursive query languages on unordered trees and data trees, among them datalog and monadic datalog. In

particular, they asked for the decidability of the query containment problem for monadic datalog on unordered labeled trees represented using the child relation and the descendant relation. The present paper gives an affirmative answer to this question, as well as an overview of the expressive power of monadic datalog on various representations of trees as logical structures.

The paper is organised as follows. Section 2 fixes the basic notation concerning unordered as well as ordered trees, and their representations as logical structures. Furthermore, it recalls the syntax and semantics, along with basic properties, of monadic datalog and monadic second-order logic. Section 3 gives details on the expressive power of monadic datalog on various kinds of tree representations. Section 4 shows that query containment, equivalence, and satisfiability of monadic datalog queries are decidable on all considered tree representations.

2 Preliminaries

We write $\mathbb{N}$ for the set of non-negative integers, and we let $\mathbb{N}_{\geqslant 1} := \mathbb{N} \setminus \{0\}$. For a set S we write 2^S to denote the power set of S, i.e., the set $\{X : X \subseteq S\}$. Throughout this paper, we let Σ be a fixed finite non-empty alphabet.

2.1 Relational Structures

In this paper, a *schema* (or, *signature*) 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* (or, *universe*) of $\mathcal{A}$, and a relation $R^{\mathcal{A}} \subseteq A^{ar(R)}$ for each relation symbol $R \in \tau$. Sometimes, it will be convenient to identify $\mathcal{A}$ with the *set of atomic facts of* $\mathcal{A}$, i.e., the set

$$\mathit{atoms}(\mathcal{A}) \quad := \quad \{\ R(a_1, \ldots, a_r) \ : \quad R \in \tau, \quad r = ar(R), \quad (a_1, \ldots, a_r) \in R^{\mathcal{A}}\ \}.$$

If τ and τ' are schemas such that $\tau \subseteq \tau'$, and $\mathcal{A}$ is a τ-structure and $\mathcal{B}$ a τ'-structure, then $\mathcal{A}$ is the *τ-reduct* of $\mathcal{B}$ (and $\mathcal{B}$ is a *τ'-expansion* of $\mathcal{A}$), if $\mathcal{A}$ and $\mathcal{B}$ have the same domain and $R^{\mathcal{A}} = R^{\mathcal{B}}$ is true for all $R \in \tau$.

2.2 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 following is true:

- There is exactly one node $\mathit{root}^T \in V^T$ with in-degree 0. This node is called the *root* of T.
- Every node $v \in V^T$ with $v \neq \mathit{root}^T$ has in-degree 1, and there is exactly one directed path from root^T to v.

As in [1], we represent unordered Σ-labeled trees T by relational structures $\mathcal{S}_u(T)$ of schema

$$\tau_u \quad := \quad \{\, \mathbf{label}_\alpha \,:\, \alpha \in \Sigma \,\} \,\cup\, \{\, \mathbf{child} \,\},$$

where **child** has arity 2 and $\mathbf{label}_\alpha$ has arity 1 (for every $\alpha \in \Sigma$), as follows:

- The domain of $\mathcal{S}_u(T)$ is the set V^T of all nodes of T,
- for each label $\alpha \in \Sigma$, $\mathbf{label}_\alpha^{\mathcal{S}_u(T)}$ consists of all nodes labeled α, i.e. $\mathbf{label}_\alpha^{\mathcal{S}_u(T)} = \{v \in V^T \,:\, \lambda^T(v) = \alpha\}$, and
- $\mathbf{child}^{\mathcal{S}_u(T)} = E^T$.

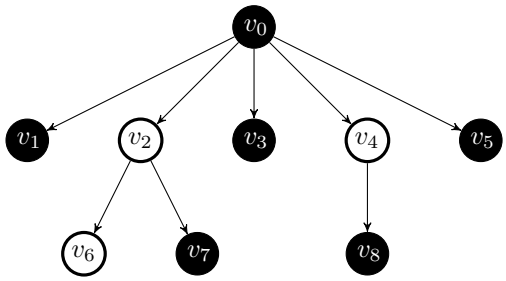

Figure 1: An example tree T labeled by symbols from $\Sigma = \{\mathit{Black}, \mathit{White}\}$.

Example 2.1. Let T be the unordered[1] Σ-labeled tree from Figure 1, for $\Sigma = \{\mathit{Black}, \mathit{White}\}$. The τ_u-structure $\mathcal{A} = \mathcal{S}_u(T)$ representing T has domain

$$A \quad = \quad \{v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\}$$

and relations

- $\mathbf{label}_{\mathit{Black}}^{\mathcal{A}} = \{v_0, v_1, v_3, v_5, v_7, v_8\}$,
- $\mathbf{label}_{\mathit{White}}^{\mathcal{A}} = \{v_2, v_4, v_6\}$,
- $\mathbf{child}^{\mathcal{A}} = \left\{ \begin{array}{l} (v_0, v_1), (v_0, v_2), (v_0, v_3), (v_0, v_4), (v_0, v_5), \\ (v_2, v_6), (v_2, v_7), (v_4, v_8) \end{array} \right\}$.

The set of atomic facts of $\mathcal{A}$ is the set $\mathit{atoms}(\mathcal{A}) =$

$$\left\{ \begin{array}{l} \mathbf{label}_{\mathit{Black}}(v_0),\ \mathbf{label}_{\mathit{Black}}(v_1),\ \mathbf{label}_{\mathit{Black}}(v_3),\ \mathbf{label}_{\mathit{Black}}(v_5), \\ \mathbf{label}_{\mathit{Black}}(v_7),\ \mathbf{label}_{\mathit{Black}}(v_8),\ \mathbf{label}_{\mathit{White}}(v_2),\ \mathbf{label}_{\mathit{White}}(v_4), \\ \mathbf{label}_{\mathit{White}}(v_6),\ \mathbf{child}(v_0, v_1),\ \mathbf{child}(v_0, v_2),\ \mathbf{child}(v_0, v_3), \\ \mathbf{child}(v_0, v_4),\ \mathbf{child}(v_0, v_5),\ \mathbf{child}(v_2, v_6),\ \mathbf{child}(v_2, v_7),\ \mathbf{child}(v_4, v_8) \end{array} \right\}.$$

⌟

[1] Note that an unordered tree does not contain any information on the relative order of the children of a node. Thus, the arrangement of children given in the picture is only one of many possibilities to draw the tree.

Sometimes, we will also consider the extended schema

$$\tau'_u \quad := \quad \tau_u \ \cup \ \{\, \mathbf{desc},\ \mathbf{are\_siblings},\ \mathbf{root},\ \mathbf{leaf} \,\}, \tag{1}$$

where **desc** and **are_siblings** are of arity 2, and **root** and **leaf** are of arity 1. The τ'_u-representation $\mathcal{S}'_u(T)$ of an unordered Σ-labeled tree T is the expansion of $\mathcal{S}_u(T)$ by the relations

- $\mathbf{desc}^{\mathcal{S}'_u(T)}$, which is the transitive (and non-reflexive) closure of E^T,
- $\mathbf{are\_siblings}^{\mathcal{S}'_u(T)}$, which consists of all tuples (u, v) of nodes such that $u \neq v$ have the same parent (i.e., there is a $w \in V^T$ such that $(w, u) \in E^T$ and $(w, v) \in E^T$).
- $\mathbf{root}^{\mathcal{S}'_u(T)}$ consists of the root node $root^T$ of T,
- $\mathbf{leaf}^{\mathcal{S}'_u(T)}$ consists of all leaves of T, i.e., all $v \in V^T$ that have out-degree 0 w.r.t. E^T.

For a set $M \subseteq \{\mathbf{desc}, \mathbf{are\_siblings}, \mathbf{root}, \mathbf{leaf}\}$ we let

$$\tau_u^M \quad := \quad \tau_u \cup M,$$

and for every Σ-labeled unordered tree T we let $\mathcal{S}_u^M(T)$ be the τ_u^M-reduct of $\mathcal{S}'_u(T)$. If M is a singleton set, we omit the curly brackets — in particular, we write $\tau_u^{\mathbf{desc}}$ instead of $\tau_u^{\{\mathbf{desc}\}}$, and $\mathcal{S}_u^{\mathbf{desc}}(T)$ instead of $\mathcal{S}_u^{\{\mathbf{desc}\}}(T)$.

2.3 Ordered Trees

An *ordered* Σ-labeled tree $T = (V^T, \lambda^T, E^T, order^T)$ consists of 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[2] of u in T.

We represent ordered Σ-labeled trees T by relational structures $\mathcal{S}_o(T)$ of schema

$$\tau_o \quad := \quad \{\, \mathbf{label}_\alpha \,:\, \alpha \in \Sigma \,\} \ \cup \ \{\, \mathbf{firstchild}, \mathbf{nextsibling} \,\},$$

where **firstchild** and **nextsibling** have arity 2 and $\mathbf{label}_\alpha$ has arity 1 (for every $\alpha \in \Sigma$) as follows:

- The domain of $\mathcal{S}_o(T)$ is the set V^T of all nodes of T,
- for each $\alpha \in \Sigma$, the relation $\mathbf{label}_\alpha^{\mathcal{S}_o(T)}$ is defined in the same way as for unordered trees,
- $\mathbf{firstchild}^{\mathcal{S}_o(T)}$ consists of all tuples (u, v) of nodes such that u is the first child of v in T (i.e., $order^T$ lists u as the first child of v),

[2] i.e., the nodes v such that $(u, v) \in E^T$

- **nextsibling**$^{\mathcal{S}_o(T)}$ consists of all tuples (v, v') of nodes such that v and v' have the same parent, i.e., there is an $u \in V^T$ such that $(u, v) \in E^T$ and $(u, v') \in E^T$, and v' is the immediate successor of v in the linear order of the children of u given by $order^T$.

Often, we will also consider the extended schema

$$\tau'_o \quad := \quad \tau_o \ \cup \ \{\, \mathbf{child},\ \mathbf{desc},\ \mathbf{root},\ \mathbf{leaf},\ \mathbf{lastsibling} \,\},$$

where **child** and **desc** have arity 2 and **root**, **leaf**, **lastsibling** have arity 1. The τ'_o-representation $\mathcal{S}'_o(T)$ of an ordered Σ-labeled tree T is the expansion of $\mathcal{S}_o(T)$ by the relations

- **child**$^{\mathcal{S}'_o(T)}$, **desc**$^{\mathcal{S}'_o(T)}$, **root**$^{\mathcal{S}'_o(T)}$, and **leaf**$^{\mathcal{S}'_o(T)}$, which are defined in the same way as for unordered trees, and
- **lastsibling**$^{\mathcal{S}'_o(T)}$, which consists of all nodes $v \neq root^T$ such that $order^T$ lists v as the last child of its parent u.

For a set $M \subseteq \{\mathbf{child},\ \mathbf{desc},\ \mathbf{root},\ \mathbf{leaf},\ \mathbf{lastsibling}\}$ we let

$$\tau_o^M \quad := \quad \tau_o \cup M,$$

and for every Σ-labeled ordered tree T we let $\mathcal{S}_o^M(T)$ be the τ_o^M-reduct of $\mathcal{S}'_u(T)$. If M is a singleton set, we omit curly brackets.

Note that in [5], Gottlob and Koch represented ordered Σ-labeled trees T by relational structures $\mathcal{S}_{GK}(T) := \mathcal{S}_o^{\{\mathbf{root},\mathbf{leaf},\mathbf{lastsibling}\}}$ of schema

$$\tau_{GK} \quad := \quad \tau_o^{\{\mathbf{root},\mathbf{leaf},\mathbf{lastsibling}\}} \quad = \quad \tau'_o \setminus \{\mathbf{child},\, \mathbf{desc}\} \quad =$$

$$\{\, \mathbf{label}_\alpha \,:\, \alpha \in \Sigma \,\} \ \cup \ \{\, \mathbf{firstchild},\ \mathbf{nextsibling},\ \mathbf{root},\ \mathbf{leaf},\ \mathbf{lastsibling} \,\}.$$

Example 2.2. Let T be the *ordered* Σ-labeled tree from Figure 1, for $\Sigma = \{Black, White\}$, where the order of the children of each node is from left to right, as depicted in the illustration. The τ_{GK}-structure $\mathcal{B} = \mathcal{S}_{GK}(T)$ representing T has domain

$$B \quad = \quad \{v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\}$$

and relations

- $\mathbf{label}^{\mathcal{B}}_{Black} = \{v_0, v_1, v_3, v_5, v_7, v_8\}$,
- $\mathbf{label}^{\mathcal{B}}_{White} = \{v_2, v_4, v_6\}$,
- $\mathbf{root}^{\mathcal{B}} = \{v_0\}$,
- $\mathbf{leaf}^{\mathcal{B}} = \{v_1, v_3, v_5, v_6, v_7, v_8\}$,
- $\mathbf{firstchild}^{\mathcal{B}} = \{\, (v_0, v_1),\ (v_2, v_6),\ (v_4, v_8) \,\}$,
- $\mathbf{nextsibling}^{\mathcal{B}} = \{\, (v_1, v_2),\ (v_2, v_3),\ (v_3, v_4),\ (v_4, v_5),\ (v_6, v_7) \,\}$,
- $\mathbf{lastsibling}^{\mathcal{B}} = \{v_5, v_7, v_8\}$.

Note that the root node of T is not included in any sibling relation. ⌟

2.4 Monadic Datalog (mDatalog)

The following definition of monadic datalog (mDatalog, for short) is basically taken from [5].

A *datalog rule* is an expression of the form

$$h \leftarrow b_1, \ldots, b_n,$$

for $n \in \mathbb{N}$, where $h, b_1, \ldots, b_n$ are called *atoms* of the rule, h is called the rule's *head*, and $b_1, \ldots, b_n$ (understood as a conjunction of atoms) is called the *body*. Each atom is of the form $P(x_1, \ldots, x_m)$ where P is a predicate of some arity $m \in \mathbb{N}_{\geqslant 1}$ and $x_1, \ldots, x_m$ are variables. Rules are required to be *safe* in the sense that all variables appearing in the head also have to appear in the body.

A *datalog program* is a finite set of datalog rules. Let $\mathcal{P}$ be a datalog program and let r be a datalog rule. We write $var(r)$ for the set of all variables occurring in the rule r, and we let $var(\mathcal{P}) := \bigcup_{r \in \mathcal{P}} var(r)$. Predicates that occur in the head of some rule of $\mathcal{P}$ are called *intensional*, whereas predicates that only occur in the body of rules of $\mathcal{P}$ are called *extensional*. We write $\mathrm{idb}(\mathcal{P})$ and $\mathrm{edb}(\mathcal{P})$ to denote the sets of intensional and extensional predicates of $\mathcal{P}$, and we say that $\mathcal{P}$ *is of schema* τ if $\mathrm{edb}(\mathcal{P}) \subseteq \tau$. A datalog program belongs to *monadic datalog* (mDatalog, for short), if all its *intensional* predicates have arity 1.

For defining the semantics of datalog, let τ be a schema, let $\mathcal{P}$ be a datalog program of schema τ, let A be a domain, and let

$$F_{\mathcal{P},A} \;:=\; \{\, R(a_1, \ldots, a_r) \,:\, R \in \tau \cup \mathrm{idb}(\mathcal{P}),\ r = ar(R),\ a_1, \ldots, a_r \in A \,\}$$

the set of all *atomic facts over* A. A *valuation* β *for* P *in* A is a function $\beta : \big(var(P) \cup A\big) \to A$ where $\beta(a) = a$ for all $a \in A$. For an atom $P(x_1, \ldots, x_m)$ occurring in a rule of $\mathcal{P}$ we let

$$\beta\big(P(x_1, \ldots, x_m)\big) \;:=\; P\big(\beta(x_1), \ldots, \beta(x_m)\big) \;\in\; F_{\mathcal{P},A}.$$

The *immediate consequence operator* $\mathcal{T}_{\mathcal{P}} : 2^{F_{\mathcal{P},A}} \to 2^{F_{\mathcal{P},A}}$ induced by the datalog program $\mathcal{P}$ on domain A maps every $C \subseteq F_{\mathcal{P},A}$ to

$$\mathcal{T}_{\mathcal{P}}(C) \;:=\; C \;\cup\; \left\{ \beta(h) \;:\; \begin{array}{l} \text{there is a rule } h \leftarrow b_1, \ldots, b_n \text{ in } \mathcal{P} \\ \text{and a valuation } \beta \text{ for } \mathcal{P} \text{ in } A \text{ such that} \\ \beta(b_1), \ldots, \beta(b_n) \in C \end{array} \right\}.$$

Clearly, $\mathcal{T}_{\mathcal{P}}$ is *monotone*, i.e., for $C \subseteq D \subseteq F_{\mathcal{P},A}$ we have $\mathcal{T}_{\mathcal{P}}(C) \subseteq \mathcal{T}_{\mathcal{P}}(D)$.

Letting $\mathcal{T}^0_{\mathcal{P}}(C) := C$ and $\mathcal{T}^{i+1}_{\mathcal{P}}(C) := \mathcal{T}_{\mathcal{P}}\big(\mathcal{T}^i_{\mathcal{P}}(C)\big)$ for all $i \in \mathbb{N}$, it is straightforward to see that

$$C = \mathcal{T}^0_{\mathcal{P}}(C) \subseteq \mathcal{T}^1_{\mathcal{P}}(C) \subseteq \cdots \subseteq \mathcal{T}^i_{\mathcal{P}}(C) \subseteq \mathcal{T}^{i+1}_{\mathcal{P}}(C) \subseteq \cdots \subseteq F_{\mathcal{P},A}.$$

For a finite domain A, the set $F_{\mathcal{P},A}$ is finite, and hence there is an $i_0 \in \mathbb{N}$ such that $\mathcal{T}^{i_0}_{\mathcal{P}}(C) = \mathcal{T}^i_{\mathcal{P}}(C)$ for all $i \geqslant i_0$. In particular, the set $\mathcal{T}^{\omega}_{\mathcal{P}}(C) := \mathcal{T}^{i_0}_{\mathcal{P}}(C)$ is a *fixpoint* of the immediate consequence operator $\mathcal{T}_{\mathcal{P}}$. By the theorem of Knaster and Tarski we know that this fixpoint is the *smallest* fixpoint of $\mathcal{T}_{\mathcal{P}}$ which contains C.

Theorem 2.3 (Knaster and Tarski [12])**.** *Let τ be a schema, let $\mathcal{P}$ be a datalog program of schema τ, and let A be a finite domain. For every $C \subseteq F_{\mathcal{P},A}$ we have*

$$\begin{aligned} \mathcal{T}_{\mathcal{P}}^{\omega}(C) &= \bigcap \{\, D \,:\, \mathcal{T}_{\mathcal{P}}(D) = D \ \textit{and}\ C \subseteq D \subseteq F_{\mathcal{P},A} \,\} \\ &= \bigcap \{\, D \,:\, \mathcal{T}_{\mathcal{P}}(D) \subseteq D \ \textit{and}\ C \subseteq D \subseteq F_{\mathcal{P},A} \,\}. \end{aligned}$$

⌟

A *k-ary (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 or extensional) predicate of arity k occurring in $\mathcal{P}$. $\mathcal{P}$ and P are called the *program* and the *query predicate* of Q. When evaluated in a finite τ-structure $\mathcal{A}$, the query Q results in the following k-ary relation over A:

$$[\![Q]\!](\mathcal{A}) \ := \ \{\, (a_1, \ldots, a_k) \in A^k \ : \ P(a_1, \ldots, a_k) \ \in \ \mathcal{T}_{\mathcal{P}}^{\omega}\big(\mathit{atoms}(\mathcal{A})\big) \,\}.$$

Unary queries are queries of arity $k = 1$.

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

$$\mathrm{edb}(\mathcal{P}) \cup \mathrm{idb}(\mathcal{P}) \cup \{x, y, z, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\} \cup \{(,), \{, \}\} \cup \{\leftarrow\} \cup \{,\}.$$

Example 2.4. Consider the schema τ_{GK} introduced in Section 2.3 for representing ordered Σ-labeled trees for $\Sigma = \{\mathit{Black},\ \mathit{White}\}$. We present a unary monadic datalog query $Q = (\mathcal{P}, \mathit{Ans})$ of schema τ_{GK} such that for every ordered Σ-labeled tree T we have

$$[\![Q]\!]\big(\mathcal{S}_{GK}(T)\big) \ = \ \begin{cases} \{\, \mathit{root}^T \,\} & \text{if the root of } T \text{ has exactly two children} \\ & \text{labeled with the symbol } \mathit{White}, \\ \emptyset & \text{otherwise.} \end{cases}$$

To this end, we let $\mathcal{P}$ consist of the following rules:

$$\begin{aligned} \mathit{Ans}(x) &\leftarrow \mathbf{root}(x),\ \mathbf{firstchild}(x, y),\ \mathit{White}_2(y) \\ \mathit{White}_2(x) &\leftarrow \mathbf{label}_{\mathit{Black}}(x),\ \mathbf{nextsibling}(x, y),\ \mathit{White}_2(y) \\ \mathit{White}_2(x) &\leftarrow \mathbf{label}_{\mathit{White}}(x),\ \mathbf{nextsibling}(x, y),\ \mathit{White}_1(y) \\ \mathit{White}_1(x) &\leftarrow \mathbf{label}_{\mathit{Black}}(x),\ \mathbf{nextsibling}(x, y),\ \mathit{White}_1(y) \\ \mathit{White}_1(x) &\leftarrow \mathbf{label}_{\mathit{White}}(x),\ \mathbf{nextsibling}(x, y),\ \mathit{White}_0(y) \\ \mathit{White}_0(x) &\leftarrow \mathbf{label}_{\mathit{Black}}(x),\ \mathbf{nextsibling}(x, y),\ \mathit{White}_0(y) \\ \mathit{White}_1(x) &\leftarrow \mathbf{label}_{\mathit{White}}(x),\ \mathbf{lastsibling}(x) \\ \mathit{White}_0(x) &\leftarrow \mathbf{label}_{\mathit{Black}}(x),\ \mathbf{lastsibling}(x) \end{aligned}$$

In particular, Q returns $\{\mathit{root}^T\}$ on the tree from Example 2.2. ⌟

Remark 2.5 (Folklore)**.** *The monotonicity of the immediate consequence operator implies that datalog queries Q of schema τ are monotone in the following sense: If $\mathcal{A}$ and $\mathcal{B}$ are τ-structures with $\mathit{atoms}(\mathcal{A}) \subseteq \mathit{atoms}(\mathcal{B})$, then $[\![Q]\!](\mathcal{A}) \subseteq [\![Q]\!](\mathcal{B})$.* ⌟

Let us point out that it is also well-known that datalog is *preserved under homomorphisms* in the following sense. A *homomorphism* from a τ-structure $\mathcal{A}$ to a τ-structure $\mathcal{B}$ is a mapping $h : A \to B$ such that for all $R \in \tau$ and all tuples $(a_1, \ldots, a_r) \in R^{\mathcal{A}}$ we have $(h(a_1), \ldots, h(a_r)) \in R^{\mathcal{B}}$. As a shorthand, for any set $S \subseteq A^k$ we let $h(S) = \{\big(h(a_1), \ldots, h(a_k)\big) : (a_1, \ldots, a_k) \in S\}$.

Lemma 2.6 (Folklore)**.** *Any k-ary datalog query Q of schema τ is preserved under homomorphisms in the following sense: If $\mathcal{A}$ and $\mathcal{B}$ are τ-structures, and h is a homomorphism from $\mathcal{A}$ to $\mathcal{B}$, then $h\big([\![Q]\!](\mathcal{A})\big) \subseteq [\![Q]\!](\mathcal{B})$.*

Proof. Let $\mathcal{A}$ and $\mathcal{B}$ be τ-structures and let $h : A \to B$ be a homomorphism from $\mathcal{A}$ to $\mathcal{B}$. Furthermore, let $Q = (\mathcal{P}, P)$ where $\mathcal{P}$ is a datalog program of schema τ. For an atomic fact $f = R(a_1, \ldots, a_r) \in F_{\mathcal{P},A}$ let $h(f) := R(h(a_1), \ldots, h(a_r))$ be the according atomic fact in $F_{\mathcal{P},B}$. Furthermore, for a set $S \subseteq F_{\mathcal{P},A}$ let $h(S) := \{h(f) : f \in S\}$ be the according subset of $F_{\mathcal{P},B}$.

First, note that by the definition of the immediate consequence operator $\mathcal{T}_{\mathcal{P}}$ it is straightforward to see that the following is true: If $C \subseteq F_{\mathcal{P},A}$ and $D \subseteq F_{\mathcal{P},B}$ such that $h(C) \subseteq D$, then $h(\mathcal{T}_{\mathcal{P}}(C)) \subseteq \mathcal{T}_{\mathcal{P}}(D)$.

Next, note that this immediately implies that the following is true: If $C \subseteq F_{\mathcal{P},A}$ and $D \subseteq F_{\mathcal{P},B}$ such that $h(C) \subseteq D$, then $h\big(\mathcal{T}^{\omega}_{\mathcal{P}}(C)\big) \subseteq \mathcal{T}^{\omega}_{\mathcal{P}}(D)$.

Finally, note that h is a homomorphism from $\mathcal{A}$ to $\mathcal{B}$, and thus $h\big(\mathit{atoms}(\mathcal{A})\big) \subseteq \mathit{atoms}(\mathcal{B})$. Consequently, $h\big(\mathcal{T}^{\omega}_{\mathcal{P}}\big(\mathit{atoms}(\mathcal{A})\big)\big) \subseteq \mathcal{T}^{\omega}_{\mathcal{P}}\big(\mathit{atoms}(\mathcal{B})\big)$. In particular, this means that $h\big([\![Q]\!](\mathcal{A})\big) \subseteq [\![Q]\!](\mathcal{B})$. □

2.5 Monadic Second-Order Logic (MSO)

The set $\mathrm{MSO}(\tau)$ of all monadic second-order formulas of schema τ is defined as usual, cf. e.g. [7]: There are two kinds of variables, namely *node variables*, denoted with lower-case letters x, y, …, x_1, x_2, … and ranging over elements of the domain, and *set variables*, denoted with upper-case letters X, Y, …, X_1, X_2, … and ranging over sets of elements of the domain.
An *atomic* $\mathrm{MSO}(\tau)$-formula is of the form

(A1) $R(x_1, \ldots, x_r)$, where $R \in \tau$, $r = \mathit{ar}(R)$, and $x_1, \ldots, x_r$ are node variables,

(A2) $x = y$, where x and y are node variables, or

(A3) $X(x)$, where x is a node variable and X is a set variable.

If x is a node variable, X a set variable, and φ and ψ are $\mathrm{MSO}(\tau)$-formulas, then

(BC) $\neg\varphi$ and $(\varphi \vee \psi)$ are $\mathrm{MSO}(\tau)$-formulas,

(Q1) $\exists x\varphi$ and $\forall x\varphi$ are $\mathrm{MSO}(\tau)$-formulas,

(Q2) $\exists X\varphi$ and $\forall X\varphi$ are $\mathrm{MSO}(\tau)$-formulas.

Quantifiers of the form (Q1) are called *first-order quantifiers*; quantifiers of the form (Q2) are called *set quantifiers*. MSO(τ)-formulas in which no set quantifier occurs, are called *first-order formulas* (FO(τ)-*formulas*, for short). The *size* $\|\varphi\|$ of a formula φ is the length of φ viewed as a string over alphabet

$$\tau \cup \{x, y, z, X, Y, Z, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\} \cup \{(,)\} \cup \{=, \neg, \vee, \exists, \forall\} \cup \{,\}.$$

As shortcuts we use the Boolean connectives $(\varphi \wedge \psi)$, $(\varphi \rightarrow \psi)$, and $(\varphi \leftrightarrow \psi)$, the statement $x \neq y$ for node variables, and the statements $X = Y$, $X \neq Y$, and $X \subseteq Y$ for set variables. Note that all these can easily be expressed in first-order logic. To improve readability of formulas, we will sometimes add or omit parentheses.

By *free*(φ) we denote the set of (node or set) variables that occur free (i.e., not within the range of a node or set quantifier) in φ. A *sentence* is a formula without free variables. We write $\varphi(x_1, \ldots, x_k, X_1, \ldots, X_\ell)$ to indicate that φ has k free node variables $x_1, \ldots, x_k$ and ℓ free set variables $X_1, \ldots, X_\ell$. For a τ-structure $\mathcal{A}$, elements $a_1, \ldots, a_k \in A$, and sets $A_1, \ldots, A_\ell \subseteq A$, we write $\mathcal{A} \models \varphi(a_1, \ldots, a_k, A_1, \ldots, A_\ell)$ to indicate that $\mathcal{A}$ satisfies the formula φ when interpreting the free occurrences of the variables $x_1, \ldots, x_k, X_1, \ldots, X_\ell$ with $a_1, \ldots, a_k, A_1, \ldots, A_\ell$. A formula $\varphi(x_1, \ldots, x_k)$ with k free node variables and no free set variable defines a k-ary query on τ-structures which, when evaluated in a τ-structure $\mathcal{A}$, results in the k-ary relation

$$[\![\varphi]\!](\mathcal{A}) \quad := \quad \{\ (a_1, \ldots, a_k) \in A^k \ : \ \mathcal{A} \models \varphi(a_1, \ldots, a_k)\ \}.$$

Example 2.7. Consider the schema τ_u introduced in Section 2.2 for representing unordered Σ-labeled trees for $\Sigma = \{\mathit{Black}, \mathit{White}\}$. We present a unary FO($\tau_u$)-query $\varphi(x)$ such that for every unordered Σ-labeled tree T we have

$$[\![\varphi]\!]\big(\mathcal{S}_u(T)\big) \quad = \quad \begin{cases} \{\ \mathit{root}^T\ \} & \text{if the root of } T \text{ has exactly two children} \\ & \text{labeled with the symbol } \mathit{White}, \\ \emptyset & \text{otherwise.} \end{cases}$$

To this end, we let $\varphi(x)$ be the following MSO(τ_u)-formula:

$$\neg\exists u\ \mathbf{child}(u, x)\ \wedge$$
$$\exists y\, \exists z \Big(y \neq z\ \wedge \mathbf{child}(x, y) \wedge \mathbf{child}(x, z) \wedge \mathbf{label}_{\mathit{White}}(y) \wedge \mathbf{label}_{\mathit{White}}(z) \wedge$$
$$\forall v \big(\ \mathbf{child}(x, v)\ \rightarrow\ (\ v = y\ \vee\ v = z\ \vee\ \neg\, \mathbf{label}_{\mathit{White}}(v)\ )\ \big)\Big)$$

⌟

A $\forall\exists$-MSO(τ)-formula is an MSO(τ)-formula of the form

$$\forall X_1 \cdots \forall X_m\ \exists x_1 \cdots \exists x_k\ \xi$$

where $m, k \in \mathbb{N}$, $X_1, \ldots, X_m$ are set variables, $x_1, \ldots, x_k$ are node variables, and ξ is a formula that does not contain any (node or set) quantifier.

It is well-known that unary monadic datalog queries can be translated into equivalent $\forall\exists$-MSO queries.

Proposition 2.8 (Folklore)**.** *Let τ be a schema. For every unary monadic datalog query $Q = (\mathcal{P}, P)$ of schema τ there is a $\forall\exists$-MSO(τ)-formula $\varphi(x)$ such that $[\![Q]\!](\mathcal{A}) = [\![\varphi]\!](\mathcal{A})$ is true for every finite τ-structure $\mathcal{A}$.*

Furthermore, there is an algorithm which computes φ from Q in time polynomial in the size of Q.

Proof. Let $\{X_1, \ldots, X_m\} = \mathrm{idb}(\mathcal{P})$ be the set of intensional predicates of $\mathcal{P}$, and w.l.o.g let $X_1 = P$. For every rule r of $\mathcal{P}$ of the form $h \leftarrow b_1, \ldots, b_n$, with $\{z_1, \ldots, z_k\} = \mathit{var}(r)$ let

$$\psi_r(X_1, \ldots, X_m) \quad := \quad \forall z_1 \cdots \forall z_k \, \big(\, (\, b_1 \wedge \cdots \wedge b_n \,) \, \rightarrow \, h \, \big).$$

Now, let $\chi(X_1, \ldots, X_m) := \bigwedge_{r \in \mathcal{P}} \psi_r(X_1, \ldots, X_m)$. Finally, let x be a node variable that does not occur in $\chi(X_1, \ldots, X_m)$ and let

$$\varphi(x) \quad := \quad \forall X_1 \cdots \forall X_m \, \big(\, \chi(X_1, \ldots, X_m) \, \rightarrow \, X_1(x) \, \big).$$

Obviously, $\varphi(x)$ is equivalent, on the class of all τ-structures, to the formula $\forall X_1 \cdots \forall X_m \big(X_1(x) \vee \neg\chi\big)$, and $\neg\chi$ is equivalent to $\bigvee_{r \in \mathcal{P}} \neg\psi_r$, while $\neg\psi_r$ is equivalent to $\exists z_1 \cdots \exists z_k \neg\big((b_1 \wedge \cdots \wedge b_n) \rightarrow h\big)$. Thus, it is straightforward to see that $\varphi(x)$ is equivalent to a $\forall\exists$-MSO(τ)-formula, and this formula can be constructed in time polynomial in the size of Q.

It remains to verify that $[\![Q]\!](\mathcal{A}) = [\![\varphi]\!](\mathcal{A})$, for every τ-structure $\mathcal{A}$. To this end, let $\mathcal{A}$ be an arbitrary τ-structure. By the construction of $\varphi(x)$ we know for $a \in A$ that

$$\begin{aligned} & a \in [\![\varphi]\!](\mathcal{A}) \\ \iff \quad & a \in X_1^{\mathcal{A}'} \ \text{ for every } \tau \cup \{X_1, \ldots, X_m\}\text{-expansion } \mathcal{A}' \text{ of } \mathcal{A} \text{ with } \mathcal{A}' \models \chi. \end{aligned}$$

Now let $C := \mathit{atoms}(\mathcal{A})$. Furthermore, consider arbitrary sets $A_1, \ldots, A_m \subseteq A$, let $\mathcal{A}'$ be the $\tau \cup \{X_1, \ldots, X_m\}$-structure obtained as the expansion of $\mathcal{A}$ by $X_i^{\mathcal{A}} := A_i$ for all $i \in \{1, \ldots, m\}$, and let $D := \mathit{atoms}(\mathcal{A}')$. Clearly, $C \subseteq D \subseteq F_{\mathcal{P},A}$. Furthermore, note that χ is constructed in such a way that the following is true:

$$\mathcal{A}' \models \chi \quad \iff \quad \mathcal{T}_{\mathcal{P}}(D) \subseteq D.$$

By the theorem of Knaster and Tarski (Theorem 2.3) we know that

$$\mathcal{T}_{\mathcal{P}}^{\omega}(C) \quad = \quad \bigcap \{ \, D \, : \, \mathcal{T}_{\mathcal{P}}(D) \subseteq D \ \text{and} \ C \subseteq D \subseteq F_{\mathcal{P},A} \, \}.$$

Thus, for $a \in A$ we have

$$\begin{aligned} & a \in [\![Q]\!](\mathcal{A}) \\ \iff \quad & X_1(a) \in \mathcal{T}_{\mathcal{P}}^{\omega}(C) \\ \iff \quad & X_1(a) \in D \ \text{ for every } D \text{ with } \mathcal{T}_{\mathcal{P}}(D) \subseteq D \text{ and } C \subseteq D \subseteq F_{\mathcal{P},A} \\ \iff \quad & a \in X_1^{\mathcal{A}'} \ \text{ for every } \tau \cup \{X_1, \ldots, X_m\}\text{-expansion } \mathcal{A}' \text{ of } \mathcal{A} \text{ with } \mathcal{A}' \models \chi \\ \iff \quad & a \in [\![\varphi]\!](\mathcal{A}). \end{aligned}$$

This completes the proof of Proposition 2.8. □

3 Expressive Power of Monadic Datalog on Trees

A unary query q on Σ-labeled (un)ordered trees assigns to each (un)ordered Σ-labeled tree T a set $q(T) \subseteq V^T$.

3.1 Expressive Power of mDatalog on Ordered Trees

Let τ be one of the schemas introduced in Section 2.3, i.e., τ is τ_o^M for some $M \subseteq \{\mathbf{child}, \mathbf{desc}, \mathbf{root}, \mathbf{leaf}, \mathbf{lastsibling}\}$. We say that a unary query q on Σ-labeled ordered trees is mDatalog(τ)-*definable* iff there is a unary monadic datalog query Q of schema τ such that for every ordered Σ-labeled tree T we have $q(T) = [\![Q]\!](\mathcal{S}_o^M(T))$. Similarly, for any subset L of MSO, q is called $L(\tau)$-*definable* iff there is an $L(\tau)$-formula $\varphi(x)$ such that for every ordered Σ-labeled tree T we have $q(T) = [\![\varphi]\!](\mathcal{S}_o^M(T))$.

Often, we will simply write $Q(T)$ instead of $[\![Q]\!](\mathcal{S}_o^M(T))$, and $\varphi(T)$ instead of $[\![\varphi]\!](\mathcal{S}_o^M(T))$.

Proposition 2.8 implies that unary queries on Σ-labeled ordered trees which are definable in mDatalog(τ), are also definable in MSO(τ). In [5] it was shown that for the particular schema $\tau = \tau_{GK}$ also the converse is true:

Theorem 3.1 (Gottlob, Koch [5])**.** *A unary query on Σ-labeled ordered trees is definable in* mDatalog(τ_{GK}) *if, and only if, it is definable in* MSO(τ_{GK})*. Furthermore, there is an algorithm which translates a given unary* mDatalog(τ_{GK})*-query into an equivalent unary* MSO(τ_{GK})*-query, and vice versa.* ⌟

In the remainder of this subsection, we point out that adding the **child** and **desc** relations won't increase the expressive power of mDatalog or MSO on ordered Σ-labeled trees, while omitting any of the relations **root**, **leaf**, or **lastsibling** will substantially decrease the expressive power of mDatalog, but not of MSO.

Fact 3.2 (Folklore)**.** *There are* MSO(τ_o)*-formulas*

$$\varphi_{\mathbf{child}}(x,y),\ \ \varphi_{\mathbf{desc}}(x,y),\ \ \varphi_{\mathbf{root}}(x),\ \ \varphi_{\mathbf{leaf}}(x),\ \ \varphi_{\mathbf{lastsibling}}(x),$$

such that for every Σ-labeled ordered tree T and all nodes a, b of T we have

$$\begin{array}{lcl}
\mathcal{S}_o(T) \models \varphi_{\mathbf{child}}(a,b) & \iff & \mathcal{S}'_o(T) \models \mathbf{child}(a,b), \\
\mathcal{S}_o(T) \models \varphi_{\mathbf{desc}}(a,b) & \iff & \mathcal{S}'_o(T) \models \mathbf{desc}(a,b), \\
\mathcal{S}_o(T) \models \varphi_{\mathbf{root}}(a) & \iff & \mathcal{S}'_o(T) \models \mathbf{root}(a), \\
\mathcal{S}_o(T) \models \varphi_{\mathbf{leaf}}(a) & \iff & \mathcal{S}'_o(T) \models \mathbf{leaf}(a), \\
\mathcal{S}_o(T) \models \varphi_{\mathbf{lastsibling}}(a) & \iff & \mathcal{S}'_o(T) \models \mathbf{lastsibling}(a).
\end{array}$$

Proof. Obviously, we can choose

$$\begin{array}{rcl}
\varphi_{\mathbf{root}}(x) & := & \neg\exists y \left(\mathbf{firstchild}(y,x) \vee \mathbf{nextsibling}(y,x) \right), \\
\varphi_{\mathbf{leaf}}(x) & := & \neg\exists y\ \mathbf{firstchild}(x,y), \\
\varphi_{\mathbf{lastsibling}}(x) & := & \neg\exists y\ \mathbf{nextsibling}(x,y).
\end{array}$$

For constructing $\varphi_{\mathbf{child}}(x,y)$ and $\varphi_{\mathbf{desc}}(x,y)$, we consider the following auxiliary formulas: Let $\varrho(x,y)$ be an arbitrary formula, let X be a set variable, and let

$$cl_{\varrho(x,y)}(X) \quad := \quad \forall x \forall y \Big(\big(X(x) \wedge \varrho(x,y) \big) \rightarrow X(y) \Big).$$

Clearly, this formula holds for a set X iff X is closed under "ϱ-successors".

In particular, the formula

$$\varphi_{\mathbf{nextsibling}^*}(x,y) \quad := \quad \forall X \Big(\big(X(x) \wedge cl_{\mathbf{nextsibling}(x,y)}(X) \big) \rightarrow X(y) \Big)$$

expresses that y is either equal to x, or it is a sibling of x which is bigger than x w.r.t. the linear order of all children of x and y's common parent. Consequently, we can choose

$$\varphi_{\mathbf{child}}(x,y) \quad := \quad \exists x' \big(\mathbf{firstchild}(x,x') \wedge \varphi_{\mathbf{nextsibling}^*}(x',y) \big).$$

Since the **desc**-relation is the transitive (and non-reflexive) closure of the **child**-relation, we can choose

$$\varphi_{\mathbf{desc}}(x,y) \quad := \quad x \neq y \wedge \forall X \Big(\big(X(x) \wedge cl_{\varphi_{\mathbf{child}}(x,y)}(X) \big) \rightarrow X(y) \Big).$$

□

In combination with Theorem 3.1 and Proposition 2.8, this leads to:

Corollary 3.3. *The following languages can express exactly the same unary queries on Σ-labeled ordered trees:*

$$\text{mDatalog}(\tau_{GK}),\ \text{mDatalog}(\tau'_o),\ \text{MSO}(\tau'_o),\ \text{MSO}(\tau_{GK}),\ \text{MSO}(\tau_o).$$

Furthermore, there is an algorithm which translates a given unary query on Σ-labeled ordered trees formulated in one of these languages into equivalent queries formulated in any of the other languages.

In particular, adding the **child** *and* **desc** *relations to τ_{GK} does not increase the expressive power of monadic datalog on Σ-labeled ordered trees.*

Proof. Since $\tau_{GK} \subseteq \tau'_o$, mDatalog($\tau_{GK}$) is at most as expressive as mDatalog(τ'_o) which, by Proposition 2.8, is at most as expressive as MSO(τ'_o).

By Fact 3.2, MSO(τ'_o) is as expressive on Σ-labeled ordered trees as MSO(τ_o) and MSO(τ_{GK}) which, by Theorem 3.1, is as expressive on Σ-labeled ordered trees as mDatalog(τ_{GK}).

Furthermore, by Proposition 2.8, Fact 3.2, and Theorem 3.1, the translation from one language to another is constructive. □

Next, we note that omitting any of the unary relations **root**, **leaf**, or **lastsibling** decreases the expressive power of monadic datalog on Σ-labeled ordered trees.

Observation 3.4. *For any relation* $\mathbf{rel} \in \{\mathbf{root}, \mathbf{leaf}, \mathbf{lastsibling}\}$*, the unary query* $q_{\mathbf{rel}}$ *with* $q_{\mathbf{rel}}(T) = \{v \in V^T : \mathcal{S}'_o(T) \models \mathbf{rel}(v)\}$ *can be expressed in* mDatalog($\{\mathbf{rel}\}$)*, but not in* mDatalog($\tau'_o \setminus \{\mathbf{rel}\}$)*.*

Proof. It is obvious that the query $q_{\mathbf{rel}}$ can be expressed in mDatalog($\{\mathbf{rel}\}$).

Let $M \subseteq \{\mathbf{child}, \mathbf{desc}, \mathbf{root}, \mathbf{leaf}, \mathbf{lastsibling}\}$ be such that $\tau_o^M = \tau'_o \setminus \{\mathbf{rel}\}$. Assume, for contradiction, that $q_{\mathbf{rel}}$ is expressed by a mDatalog(τ_o^M)-query $Q = (\mathcal{P}, P)$.

First, consider the case where $\mathbf{rel} = \mathbf{root}$. Let T_0 be the tree consisting of a single node v labeled $\alpha \in \Sigma$, and let T_1 be the tree consisting of two nodes u, v, both labeled α, such that v is the unique child of u. Since $\tau_o^M = \tau'_o \setminus \{\mathbf{root}\}$, we have

$$\begin{aligned} atoms\big(\mathcal{S}_o^M(T_0)\big) &= \{\, \mathbf{label}_\alpha(v),\ \mathbf{leaf}(v)\,\}, \quad \text{and} \\ atoms\big(\mathcal{S}_o^M(T_1)\big) &= atoms\big(\mathcal{S}_o^M(T_0)\big) \cup \left\{ \begin{array}{l} \mathbf{label}_\alpha(u),\ \mathbf{firstchild}(u,v), \\ \mathbf{lastsibling}(v),\ \mathbf{child}(u,v), \\ \mathbf{desc}(u,v) \end{array} \right\}. \end{aligned}$$

I.e., $atoms(\mathcal{S}_o^M(T_0)) \subseteq atoms(\mathcal{S}_o^M(T_1))$ and thus, due to the monotonicity stated in Remark 2.5, we have $[\![Q]\!](\mathcal{S}_o^M(T_0)) \subseteq [\![Q]\!](\mathcal{S}_o^M(T_1))$. This contradicts the fact that $v \in q_{\mathbf{root}}(T_0) = [\![Q]\!](\mathcal{S}_o^M(T_0))$ but $v \notin q_{\mathbf{root}}(T_1) = [\![Q]\!](\mathcal{S}_o^M(T_1))$.

Next, consider the case where $\mathbf{rel} = \mathbf{leaf}$, and let T_0 be the tree consisting of a single node v labeled $\alpha \in \Sigma$, and let T'_1 be the tree consisting of two nodes v and w, both labeled α, such that w is the unique child of v. Since $\tau_o^M = \tau'_o \setminus \{\mathbf{leaf}\}$, it is straightforward to see that $atoms(\mathcal{S}_o^M(T_0)) \subseteq atoms(\mathcal{S}_o^M(T'_1))$. By monotonicity, we have that $[\![Q]\!](\mathcal{S}_o^M(T_0)) \subseteq [\![Q]\!](\mathcal{S}_o^M(T'_1))$, contradicting the fact that $v \in q_{\mathbf{leaf}}(T_0) = [\![Q]\!](\mathcal{S}_o^M(T_0))$ but $v \notin q_{\mathbf{leaf}}(T'_1) = [\![Q]\!](\mathcal{S}_o^M(T'_1))$.

Finally, consider the case where $\mathbf{rel} = \mathbf{lastsibling}$. Let T_1 be the tree consisting of two nodes u, v, both labeled α, such that v is the unique child of u. Let T_2 be the tree consisting of three nodes u, v, w, all labeled α, such that v and w are the first and the second child of u. Since $\tau_o^M = \tau'_o \setminus \{\mathbf{lastsibling}\}$, it is straightforward to see that $atoms(\mathcal{S}_o^M(T_1)) \subseteq atoms(\mathcal{S}_o^M(T_2))$. By monotonicity, we have $[\![Q]\!](\mathcal{S}_o^M(T_1)) \subseteq [\![Q]\!](\mathcal{S}_o^M(T_2))$, contradicting the fact that $v \in q_{\mathbf{lastsibling}}(T_1) = [\![Q]\!](\mathcal{S}_o^M(T_1))$ but $v \notin q_{\mathbf{lastsibling}}(T_2) = [\![Q]\!](\mathcal{S}_o^M(T_2))$. □

3.2 Expressive Power of mDatalog on Unordered Trees

Let τ be one of the schemas introduced in Section 2.2, i.e., τ is τ_u^M for some $M \subseteq \{\mathbf{desc}, \mathbf{are\_siblings}, \mathbf{root}, \mathbf{leaf}\}$. We say that a unary query q on Σ-labeled unordered trees is mDatalog(τ)-*definable* iff there is a unary monadic datalog query Q of schema τ such that for every unordered Σ-labeled tree T we have $q(T) = [\![Q]\!](\mathcal{S}_u^M(T))$. Similarly, for any subset L of MSO, q is called $L(\tau)$-*definable* iff there is an $L(\tau)$-formula $\varphi(x)$ such that for every unordered Σ-labeled tree T we have $q(T) = [\![\varphi]\!](\mathcal{S}_u^M(T))$.

Often, we will simply write $Q(T)$ instead of $[\![Q]\!](\mathcal{S}_u^M(T))$, and $\varphi(T)$ instead of $[\![\varphi]\!](\mathcal{S}_u^M(T))$.

Proposition 2.8 implies that unary queries on Σ-labeled unordered trees which are definable in mDatalog(τ), are also definable in MSO(τ). It is straightforward to see that MSO(τ_u) can express all the relations present in τ'_u:

Fact 3.5 (Folklore)**.** *There are* MSO(τ_u)*-formulas*

$$\varphi_{\mathbf{desc}}(x,y), \quad \varphi_{\mathbf{are\_siblings}}(x,y), \quad \varphi_{\mathbf{root}}(x), \quad \varphi_{\mathbf{leaf}}(x)$$

such that for every Σ*-labeled unordered tree* T *and all nodes* a, b *of* T *we have*

$$\begin{array}{lcl}
\mathcal{S}_u(T) \models \varphi_{\mathbf{desc}}(a,b) & \iff & \mathcal{S}'_u(T) \models \mathbf{desc}(a,b), \\
\mathcal{S}_u(T) \models \varphi_{\mathbf{are_siblings}}(a,b) & \iff & \mathcal{S}'_u(T) \models \mathbf{are_siblings}(a,b), \\
\mathcal{S}_u(T) \models \varphi_{\mathbf{root}}(a) & \iff & \mathcal{S}'_u(T) \models \mathbf{root}(a), \\
\mathcal{S}_u(T) \models \varphi_{\mathbf{leaf}}(a) & \iff & \mathcal{S}'_u(T) \models \mathbf{leaf}(a).
\end{array}$$

Proof. Obviously, we can choose

$$\begin{array}{rcl}
\varphi_{\mathbf{root}}(x) & := & \neg\, \exists y\ \mathbf{child}(y,x), \\
\varphi_{\mathbf{leaf}}(x) & := & \neg\, \exists y\ \mathbf{child}(x,y), \\
\varphi_{\mathbf{are_siblings}}(x,y) & := & x \neq y\ \wedge\ \exists u \big(\ \mathbf{child}(u,x)\ \wedge\ \mathbf{child}(u,y)\ \big).
\end{array}$$

For constructing $\varphi_{\mathbf{desc}}(x,y)$, we consider the following auxiliary formula: Let $\varrho(x,y)$ be an arbitrary formula, let X be a set variable, and let

$$cl_{\varrho(x,y)}(X) \quad := \quad \forall x \forall y \Big(\big(X(x) \wedge \varrho(x,y) \big) \ \rightarrow\ X(y) \Big).$$

Clearly, this formula holds for a set X iff X is closed under "ϱ-successors".

In particular, the formula

$$\varphi_{\mathbf{child}^*}(x,y) \quad := \quad \forall X \Big(\big(X(x)\ \wedge\ cl_{\mathbf{child}(x,y)}(X) \big)\ \rightarrow\ X(y) \Big)$$

expresses that y is either equal to x, or it is a descendant of x. Thus, we can choose

$$\varphi_{\mathbf{desc}}(x,y) \quad := \quad x \neq y\ \wedge\ \varphi_{\mathbf{child}^*}(x,y).$$

□

However, unlike in the case of ordered trees, mDatalog(τ'_u) cannot express all unary queries expressible in MSO(τ_u), as the following observation shows.

Observation 3.6. *The unary query* q_{two} *with*

$$q_{two}(T) \quad = \quad \{v \in V^T\ :\ v \textit{ has exactly two children labeled } \alpha\}$$

is expressible in MSO(τ_u)*, but not in* mDatalog(τ'_u)*.*

Proof. It is obvious that the query q_{two} is defined by the MSO(τ_u)-formula $\psi(x) :=$

$$\exists y_1 \exists y_2 \Big(\mathbf{child}(x, y_1) \wedge \mathbf{child}(x, y_2) \wedge \mathbf{label}_\alpha(y_1) \wedge \mathbf{label}_\alpha(y_2) \wedge y_1 \neq y_2 \wedge \forall z \big((\, \mathbf{child}(x, z) \wedge \mathbf{label}_\alpha(z) \,) \rightarrow (\, z = y_1 \vee z = y_2 \,) \big) \Big).$$

For contradiction, assume that q_{two} is expressed by a mDatalog(τ'_u)-query $Q = (\mathcal{P}, P)$. Let T_2 be the Σ-labeled unordered tree consisting of three nodes u, v_1, v_2, all labeled α, such that v_1 and v_2 are children of u. Furthermore, let T_3 be the tree consisting of four nodes u, v_1, v_2, v_3, all labeled α, such that v_1, v_2, v_3 are children of u. Since

$$\tau'_u \;=\; \{\mathbf{label}_\alpha \,:\, \alpha \in \Sigma\} \;\cup\; \{\, \mathbf{child}, \mathbf{desc}, \mathbf{are\_siblings}, \mathbf{root}, \mathbf{leaf} \,\},$$

it is straightforward to see that $atoms(\mathcal{S}'_u(T_2)) \subseteq atoms(\mathcal{S}'_u(T_3))$. Thus, due to the monotonicity stated in Remark 2.5, we have $[\![Q]\!](\mathcal{S}'_u(T_2)) \subseteq [\![Q]\!](\mathcal{S}'_u(T_3))$. This contradicts the fact that $u \in q_{two}(T_2) = [\![Q]\!](\mathcal{S}'_u(T_2))$ but $u \notin q_{two}(T_3) = [\![Q]\!](\mathcal{S}'_u(T_3))$. □

Next, we note that omitting any of the relations **root**, **leaf**, or **are_siblings** further decreases the expressive power of monadic datalog on Σ-labeled unordered trees.

Observation 3.7. *(a) For any relation* $\mathbf{rel} \in \{\mathbf{root}, \mathbf{leaf}\}$*, the query* $q_{\mathbf{rel}}$ *with* $q_{\mathbf{rel}}(T) = \{v \in V^T \,:\, \mathcal{S}'_u(T) \models \mathbf{rel}(v)\}$ *can be expressed in* mDatalog($\{\mathbf{rel}\}$)*, but not in* mDatalog($\tau'_u \setminus \mathbf{rel}$)*.*

(b) The query q_{sib} *with* $q_{sib}(T) = \{v \in V^T \,:\, v$ *has at least one sibling*$\}$*, for all* Σ*-labeled unordered trees* T*, can be expressed in* mDatalog($\{\mathbf{are\_siblings}\}$)*, but not in* mDatalog($\tau'_u \setminus \{\mathbf{are\_siblings}\}$)*.*

Proof. The proof of (a) is analogous to the according parts of the proof of Observation 3.4.

For the proof of (b), first note that q_{sib} is expressed by the unary monadic datalog query $Q = (\mathcal{P}, P)$ where Q consists of the single rule

$$P(x) \leftarrow \mathbf{are\_siblings}(x, y).$$

Now let $M = \{\mathbf{desc}, \mathbf{root}, \mathbf{leaf}\}$, i.e., $\tau_u^M = \tau'_u \setminus \{\mathbf{are\_siblings}\}$. Assume, for contradiction, that q_{sib} is expressed by a unary mDatalog(τ_u^M)-query $Q = (\mathcal{P}, P)$. We will conclude the proof by using Lemma 2.6, stating that datalog queries are preserved under homomorphisms.

Let T_2 be the Σ-labeled unordered tree consisting of three nodes a, a_1, a_2, all labeled α, such that a_1 and a_2 are children of a. Furthermore, let T_1 be the tree consisting of two nodes b, b_1, both labeled α, such that b_1 is the unique child of b. Let $\mathcal{A} := \mathcal{S}_u^M(T_2)$ and $\mathcal{B} := \mathcal{S}_u^M(T_1)$.

Consider the mapping $h : A \rightarrow B$ with $h(a) = b$ and $h(a_1) = h(a_2) = b_1$. It is not difficult to see that h is a homomorphism from $\mathcal{A}$ to $\mathcal{B}$, since

- $\mathbf{label}^{\mathcal{A}}_{\alpha} = \{a, a_1, a_2\}$ and $\mathbf{label}^{\mathcal{B}}_{\alpha} = \{b, b_1\}$
- $\mathbf{label}^{\mathcal{A}}_{\alpha'} = \emptyset = \mathbf{label}^{\mathcal{B}}_{\alpha'}$, for all $\alpha' \in \Sigma$ with $\alpha' \neq \alpha$
- $\mathbf{child}^{\mathcal{A}} = \{(a, a_1), (a, a_2)\}$ and $\mathbf{child}^{\mathcal{B}} = \{(b, b_1)\}$
- $\mathbf{desc}^{\mathcal{A}} = \mathbf{child}^{\mathcal{A}}$ and $\mathbf{desc}^{B} = \mathbf{child}^{B}$
- $\mathbf{root}^{\mathcal{A}} = \{a\}$ and $\mathbf{root}^{\mathcal{B}} = \{b\}$
- $\mathbf{leaf}^{\mathcal{A}} = \{a_1, a_2\}$ and $\mathbf{leaf}^{\mathcal{B}} = \{b_1\}$.

From Lemma 2.6 we obtain that $h\big([\![Q]\!](\mathcal{A})\big) \subseteq [\![Q]\!](\mathcal{B})$. This contradicts the fact that $a_1 \in q_{sib}(T_2) = [\![Q]\!](\mathcal{A})$, but $h(a_1) = b_1 \notin q_{sib}(T_1) = [\![Q]\!](\mathcal{B})$. □

In summary, we immediately obtain the following:

Corollary 3.8. *(a)* $\mathrm{MSO}(\tau_u)$ *can express exactly the same unary queries on* Σ*-labeled unordered trees as* $\mathrm{MSO}(\tau'_u)$*; and there is a polynomial time algorithm which translates a given unary* $\mathrm{MSO}(\tau'_u)$*-query on* Σ*-labeled unordered trees into an equivalent* $\mathrm{MSO}(\tau_u)$*-query.*

Furthermore, both languages are capable of expressing strictly more unary queries on Σ*-labeled unordered trees than* $\mathrm{mDatalog}(\tau'_u)$*.*

(b) Omitting any of the relations **root**, **leaf**, **are_siblings** *strictly decreases the expressive power of unary* $\mathrm{mDatalog}(\tau'_u)$*-queries on* Σ*-labeled unordered trees.* ⌟

4 Query containment, Equivalence, and Satisfiability for Monadic Datalog on Trees

Query containment, equivalence, and satisfiability of queries are important problems concerning query optimisation and static analysis of queries.

4.1 Query Containment for mDatalog on Trees

Let τ be one of the schemas introduced in Section 2.2 or Section 2.3, and let $\mathcal{S}(T)$ the corresponding τ-structure representing the tree T.

For two queries Q_1 and Q_2 of schema τ, we write $Q_1 \subseteq Q_2$ (and say that Q_1 is included in Q_2 on trees) to indicate that for every Σ-labeled tree T we have $[\![Q_1]\!](\mathcal{S}(T)) \subseteq [\![Q_2]\!](\mathcal{S}(T))$. Accordingly, we write $Q_1 \not\subseteq Q_2$ to indicate that $Q_1 \subseteq Q_2$ does not hold.

An important task for query optimisation and static analysis is the *query containment problem*, defined as follows:

The QCP for unary mDatalog(τ)-queries on (un)ordered Σ-labeled trees	
Input: Two unary mDatalog(τ)-queries Q_1 and Q_2.	
Output:	`Yes`, if $Q_1 \subseteq Q_2$, `No`, otherwise.

For *ordered* Σ-labeled trees, the following is known:

Theorem 4.1 (Gottlob, Koch [5])**.**
The QCP for unary mDatalog(τ_{GK})*-queries on ordered* Σ*-labeled trees is decidable and* EXPTIME*-hard.* ⌟

Using Corollary 3.3 and the fact that $\tau_{GK} \subseteq \tau'_o$, this immediately leads to:

Theorem 4.2. *The QCP for unary* mDatalog(τ'_o)*-queries on ordered* Σ*-labeled trees is decidable and* EXPTIME*-hard.* ⌟

To obtain decidability also for the case of *unordered* Σ-labeled trees, we can use the following result:

Theorem 4.3 (Seese [11])**.** *The problem*

Satisfiability of MSO(τ_u)-sentences on unordered Σ-labeled trees
Input: An MSO(τ_u)*-sentence* φ*.*
Question: Does there exist an unordered Σ*-labeled (finite) tree* T *such that* $\mathcal{S}_u(T) \models \varphi$*?*

is decidable. ⌟

Combining this with Proposition 2.8 and Fact 3.5, we obtain:

Theorem 4.4. *The QCP for unary* mDatalog(τ'_u)*-queries on unordered* Σ*-labeled trees is decidable.*

Proof. An algorithm for deciding the QCP for unary mDatalog(τ'_u)-queries on unordered Σ-labeled trees can proceed as follows:

On input of two unary mDatalog(τ'_u)-queries Q_1 and Q_2, first use the algorithm from Proposition 2.8 to construct two MSO(τ'_u)-formulas $\varphi_1(x)$ and $\varphi_2(x)$ such that, for each $i \in \{1, 2\}$, the formula $\varphi_i(x)$ defines the same unary query on Σ-labeled unordered trees as Q_i.

Afterwards, use Fact 3.5 to translate the MSO(τ'_u)-formulas $\varphi_1(x)$ and $\varphi_2(x)$ into MSO(τ_u)-formulas $\psi_1(x)$ and $\psi_2(x)$, which are equivalent to $\varphi_1(x)$ and $\varphi_2(x)$ on Σ-labeled unordered trees.

Finally, let

$$\varphi \quad := \quad \exists x \, \big(\psi_1(x) \wedge \neg \psi_2(x) \big),$$

and use the algorithm provided by Theorem 4.3 to decide whether there is an unordered Σ-labeled tree T such that $\mathcal{S}_u(T) \models \varphi$. Output "`No`" if this algorithm outputs "`Yes`", and output "`Yes`" otherwise.

To verify that this algorithm produces the correct answer, note that for every Σ-labeled unordered tree T, the following is true:

$$\begin{array}{ll} & \mathcal{S}_u(T) \models \varphi \\ \iff & \text{there is a node } a \text{ of } T \text{ with } \mathcal{S}'_u(T) \models \psi_1(a) \text{ and } \mathcal{S}'_u(T) \not\models \psi_2(a) \\ \iff & \text{there is a node } a \text{ of } T \text{ with } a \in [\![Q_1]\!](\mathcal{S}'_u(T)) \text{ and } a \notin [\![Q_2]\!](\mathcal{S}'_u(T)) \\ \iff & [\![Q_1]\!](\mathcal{S}'_u(T)) \not\subseteq [\![Q_2]\!](\mathcal{S}'_u(T)). \end{array}$$

Thus, the MSO(τ_u)-sentence φ is satisfiable on unordered Σ-labeled trees if, and only if, $Q_1 \not\subseteq Q_2$. □

4.2 Equivalence for mDatalog on Trees

Let τ be one of the schemas introduced in Section 2.2 or Section 2.3, and let $\mathcal{S}(T)$ the corresponding τ-structure representing the tree T.

For two queries Q_1 and Q_2 of schema τ, we write $Q_1 \equiv Q_2$ (and say that Q_1 is equivalent to Q_2 on trees) to indicate that for every Σ-labeled tree T we have $[\![Q_1]\!](\mathcal{S}(T)) = [\![Q_2]\!](\mathcal{S}(T))$. Accordingly, we write $Q_1 \not\equiv Q_2$ to indicate that $Q_1 \equiv Q_2$ does not hold. We consider the following decision problem.

The Equivalence Problem for unary mDatalog(τ)-queries on Σ-labeled (un)ordered trees

Input: Two unary mDatalog(τ)-queries Q_1 and Q_2.

Output: `Yes`, if $Q_1 \equiv Q_2$,
`No`, otherwise.

By definition, we have $Q_1 \equiv Q_2$ if, and only if, $Q_1 \subseteq Q_2$ and $Q_2 \subseteq Q_1$. Thus, the decidability of the query containment problem for mDatalog stated in Theorem 4.2 and Theorem 4.4, immediately leads to the following.

Corollary 4.5.

(a) The equivalence problem for unary mDatalog(τ'_o)*-queries on Σ-labeled ordered trees is decidable.*

(b) The equivalence problem for unary mDatalog(τ'_u)*-queries on Σ-labeled unordered trees is decidable.* ⌟

4.3 Satisfiability of mDatalog on Trees

Let τ be one of the schemas introduced in Section 2.2 or Section 2.3, and let $\mathcal{S}(T)$ the corresponding τ-structure representing the tree T.

A query Q of schema τ is called *satisfiable on trees* if there is a Σ-labeled (un)ordered tree T such that $[\![Q]\!](\mathcal{S}(T)) \neq \emptyset$.

Example 4.6. There exists a unary mDatalog(τ)-query $Q_{unsat} = (\mathcal{P}_{unsat}, P_{unsat})$ which is *not* satisfiable on trees.
For example, for $\tau = \tau_u$ the $\mathcal{P}_{unsat}$ can be chosen to consist of the single rule

$$P_{unsat}(x) \quad \leftarrow \quad \mathbf{child}(x, x)$$

and for $\tau = \tau_o$ the following rule can be chosen

$$P_{unsat}(x) \quad \leftarrow \quad \mathbf{firstchild}(x, x)$$

since in trees no node can be its own (first)child. ⌟

We consider the following decision problem.

The Satisfiability Problem for unary mDatalog(τ)-queries on Σ-labeled (un)ordered trees		
Input:	A unary mDatalog(τ)-query Q.	
Output:	`Yes`,	if Q is satisfiable on trees,
	`No`,	otherwise.

Corollary 4.5, together with Example 4.6, leads to the following.

Corollary 4.7. *(a) The satisfiability problem for unary* mDatalog(τ'_o)*-queries on Σ-labeled ordered trees is decidable.*

(b) The satisfiability problem for unary mDatalog(τ'_u)*-queries on Σ-labeled unordered trees is decidable.*

Proof. Let Q be the input query for which we want to decide whether or not it is satisfiable on trees. Let Q_{unsat} be the unsatisfiable query from Example 4.6.

It is straightforward to see that $Q \equiv Q_{unsat}$ if, and only if, Q is *not* satisfiable on trees. Thus, we can use the algorithms for deciding equivalence of queries on trees (provided by Corollary 4.5) to decide whether or not Q is satisfiable on trees. □

References

[1] Serge Abiteboul, Pierre Bourhis, Anca Muscholl, and Zhilin Wu, *Recursive queries on trees and data trees*, Proceedings of the 16th International Conference on Database Theory (ICDT'13), ACM, 2013, pp. 93–104.

[2] Henrik Björklund, Wim Martens, and Thomas Schwentick, *Conjunctive query containment over trees*, J. Comput. Syst. Sci. **77** (2011), no. 3, 450–472.

[3] Mikolaj Bojanczyk, Anca Muscholl, Thomas Schwentick, and Luc Segoufin, *Two-variable logic on data trees and xml reasoning*, J. ACM **56** (2009), no. 3.

[4] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi, *Tree automata techniques and applications*, Available on: `http://www.grappa.univ-lille3.fr/tata`, 2008, release November, 18th 2008.

[5] G. Gottlob and C. Koch, *Monadic datalog and the expressive power of languages for web information extraction*, J. ACM **51** (2004), no. 1, pp. 74–113.

[6] Stephan Kepser, *A landscape of logics for finite unordered unranked trees*, Formal Grammar 2008 (Philippe de Groote, Laura Kallmeyer, Gerald Penn, and Giorgio Satta, eds.), CSLI Publications, 2008.

[7] Leonid Libkin, *Elements of finite model theory*, Springer-Verlag, 2004.

[8] ———, *Logics for unranked trees: An overview*, Logical Methods in Computer Science **2** (2006), no. 3.

[9] Frank Neven, *Automata, logic, and xml*, Proc. 16th International Workshop, CSL 2002, 11th Annual Conference of the EACSL (CSL'02), Lecture Notes in Computer Science, vol. 2471, Springer-Verlag, 2002, pp. 2–26.

[10] Frank Neven and Thomas Schwentick, *Query automata over finite trees*, Theor. Comput. Sci. **275** (2002), no. 1-2, 633–674.

[11] D. Seese, *The structure of the models of decidable monadic theories of graphs*, Annals of Pure and Applied Logic **53** (1991), no. 2, 169–195.

[12] Alfred Tarski, *A lattice-theoretical fixpoint theorem and its applications*, Pacific Journal of Mathematics **5** (1955), no. 2, 285–309.

[13] Wolfgang Thomas, *Languages, automata, and logic*, Handbook of Formal Languages, vol. 3, Springer-Verlag, 1997, pp. 389–455.