Recursive queries on trees and data trees
Serge Abiteboul, Pierre Bourhis, Anca Muscholl, Zhilin Wu

To cite this version:
Serge Abiteboul, Pierre Bourhis, Anca Muscholl, Zhilin Wu. Recursive queries on trees and data trees. Proceedings of the 16th International Conference on Database Theory, Mar 2013, Genoa, Italy. hal-00809297

HAL Id: hal-00809297
https://inria.hal.science/hal-00809297
Submitted on 8 Apr 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Recursive queries on trees and data trees

Serge Abiteboul
INRIA Saclay & ENS Cachan
Serge.Abiteboul@inria.fr

Pierre Bourhis
Oxford University
bourhis@comlab.ox.ac.uk

Anca Muscholl
LaBRI, University of Bordeaux
anca@labri.fr

Zhilin Wu
State Key Lab. of Comp. Sci., Inst. of Software, Chinese Academy of Sciences
wuzl@ios.ac.cn

ABSTRACT

The analysis of datalog programs over relational structures has been studied in depth, most notably the problem of containment. The analysis problems that have been considered were shown to be undecidable with the exception of (i) containment of arbitrary programs in nonrecursive ones, (ii) containment of monadic programs, and (iii) emptiness. In this paper, we are concerned with a much less studied problem, the analysis of datalog programs over data trees. We show that the analysis of datalog programs is more complex for data trees than for arbitrary structures. In particular, we prove that the three aforementioned problems are undecidable for data trees. But in practice, data trees (e.g., XML trees) are often of bounded depth. We prove that all three problems are decidable over bounded depth data trees.

Another contribution of the paper is the study of a new form of automata called pattern automata, that are essentially equivalent to linear datalog programs. We use pattern automata to show that the emptiness problem for linear monadic datalog programs with data value inequalities is decidable over arbitrary data trees.

1. INTRODUCTION

In this paper, we study recursive queries over data trees. The trees we consider are unordered, unranked, labelled trees with data from an infinite domain. For specifying recursive queries, we use datalog on a relational representation of the trees. We show that the analysis of datalog programs is more complex for data trees than for arbitrary structures. We show that for bounded depth data trees, one can solve

analysis problems that are undecidable for data trees in general. In our study, we introduce and use pattern automata, that correspond to linear datalog programs.

Following [5, 12], we describe a data tree by a relational structure as follows. A tree is described by relations child and desc (for descendant) between nodes. Each node has a label from a finite fixed alphabet of labels and a value from an infinite set of data values. For each label α, a unary relation gives all the nodes with that particular label. The “data” is captured by a relation ∼, with n1 ∼ n2 if the two nodes n1, n2 have the same data value.

The analysis of datalog programs has been studied in depth, most notably the problem of containment. The problems that have been considered were shown to be undecidable with the exception of (i) containment of arbitrary programs in nonrecursive ones, (ii) containment of monadic programs, and (iii) emptiness. We show that the analysis of datalog programs is more complex for data trees than for arbitrary structures. In particular, we prove that the three aforementioned problems are undecidable for data trees.

We show that emptiness of a datalog program over trees is undecidable even for trees without data, and even for words. This illustrates the complexity introduced by the constraints of the tree structure. On the positive side, emptiness over trees becomes decidable for linear monadic programs for trees with data, and even for queries with inequalities. This shows notably the power of arity 2 that allows keeping track simultaneously of two computations. Whether decidability also holds for non-linear monadic programs remains open.

In practice, data trees (like in XML) are often of bounded depth. We prove that (i), (ii) and (iii) are decidable over bounded depth data trees. From a practical viewpoint, these are probably the most meaningful results of the paper, in spite of the double exponential lower bounds. This shows that, when attempting to analyse datalog programs over trees (e.g. for optimization), assuming the boundedness of trees may greatly simplify the problem.

Another contribution of the paper is the study of a new form of automata called pattern automata, that are essentially equivalent to linear datalog programs. We use pattern automata to show that the emptiness problem for linear monadic datalog programs with data value inequalities is decidable over arbitrary data trees.

2. DEFINITIONS

Consider a finite alphabet Σ of labels and an infinite set N of nodes. The structures we consider here are finite, labeled, unranked and unordered trees. To represent trees, we use the
relational schema $S = \{ \text{child}, \text{desc} \} \cup \Sigma$. The representation $I$ of a tree $t$ is defined by the following (with $n, n_1, n_2 \in \mathbb{N}$):

- $\text{child}(n_1, n_2)$ if $n_2$ is a child of $n_1$;
- $\text{desc}(n_1, n_2)$ if $n_2$ is a strict descendant of $n_1$;
- For each $\sigma$ in $\Sigma$, $\sigma(n)$ if the label of $n$ is $\sigma$.

Observe that the instances satisfy some particular constraints because they represent trees. For instance,

- The second attribute of the relation $\text{child}$ is a primary key (each node has at most one parent)
- There exists a single value $n$ that occurs in the first column of $\text{child}$ and not the second (the unique root).
- $\text{desc}$ is the (nonreflexive) transitive closure of $\text{child}$.
- The relations $\sigma \in \Sigma$ partition the set of nodes.

Data trees. In this paper, the focus is on queries over data trees defined as follows.

Consider an infinite set $\mathbb{D}$ of data values. A data tree $t$ is a tree such that to each of its nodes is also associated some value of $\mathbb{D}$. To represent data trees, we use the relational schema $S^\sim = \{ \text{child}, \text{desc}, \sim \} \cup \Sigma$.

The representation $I$ of a data tree $t$ is defined by

- $\text{child}, \text{desc}$ and each $\sigma$ in $\Sigma$ are as in the definition of trees, and
- $n_1 \sim n_2$ if the nodes $n_1, n_2$ in $t$ have the same data value.

Note that $\sim$ is an equivalence relation. Data words can be defined as the special case of data trees where each non-leaf node has exactly one child.

Datalog and UCQs. We assume that the reader is familiar with datalog. We recall briefly some definitions next. A datalog program consists of a finite set of rules of the form:

$$R(\bar{x}) \leftarrow R_1(\bar{x}_1), \ldots, R_k(\bar{x}_k)$$

such that $k \geq 0$, $\bar{x}$ and each $\bar{x}_i$ are vectors of variables and constants and each variable occurring in the head $R$ of the rule also occurs in the body. A datalog program is defined over a database schema $Sch$ and predicates from $Sch$ are called extensional, the other predicates are called intensional. Only intensional predicates can occur in rule heads. A distinguished intensional predicate is called the goal. A datalog program defines a mapping from database instances over $Sch$ to relational instances over $goal$.

We also consider some restrictions. A datalog program is:

- linear if each rule body includes at most one intensional atom;
- $k$-datalog if the arity of each intensional predicate is at most $k$, and monadic if it is at most one, i.e., it is 1-datalog ($k$ is also called the arity of the datalog program); and
- Boolean if the arity of the goal predicate is zero.

The answer of a datalog program $P$ over an instance $I$, denoted by $P(I)$, is the set of tuples appearing in the goal predicate after having filled the intensional predicates by using recursively the rules of the datalog program until a fixpoint is reached.

In this paper, we consider datalog programs over the schema $S^\sim$, where $S^\sim = S^\sim \cup \{ \neq, \sim \}$ is the extended schema for data trees, with $\neq$ and $\sim$ interpreted respectively as node inequality and data value inequality. We use the notation $\text{datalog}^\sim$ to denote the datalog programs over the schema $S^\sim$.

A conjunctive query (CQ) is an existential formula of the form $\exists x_1 \cdots \exists x_k \varphi$, where $\varphi$ is a conjunction of positive atomic formulas over the given signature. We will consider unions of conjunctive queries, UCQs for short. A UCQ is Boolean if it has no free variable.

To conclude this section, we briefly recall some known decidability results on the containment problem for datalog programs.

First, we consider arbitrary structures, then trees. For arbitrary structures, the following result (that will be used in Section 4) is known:

**Theorem 2.1.** [9]

- The problem of containment of a datalog program in a UCQ for relational structures is $2\text{EXPTIME}$-complete.
- The problem of containment of a linear datalog program in a UCQ for relational structures is $\text{EXPSPACE}$-complete.

The following result for trees without data and monadic datalog programs not using the descendant relation is also known:

**Theorem 2.2.** [14] Let $P$ be a monadic datalog program over the schema $S \setminus \{ \text{desc} \} = \{ \text{child} \} \cup \Sigma$. Then there exists an unranked query tree automaton $A_P$ such that for each tree $t$ over $\Sigma$ and the representation $I$ of $t$, $P(I) = A_P(t)$.

Using the previous result, we observe:

**Corollary 2.3.** The containment problem for monadic datalog programs over trees not using the relation desc (i.e., over the schema $\{ \text{child} \} \cup \Sigma$) is decidable.

The containment problem for monadic datalog programs over trees remains open. We will see in the next section that it is undecidable over data trees. And in the following one, that the problem is decidable for bounded-depth data trees.

3. UNDECIDABILITY RESULTS

The containment problem for datalog programs is known to be undecidable for arbitrary structures [1]. However, it has been shown to be decidable for two important natural restrictions: (i) for the containment of a datalog program in a non-recursive program [9], and (ii) for monadic datalog programs [10]. These decidability results hold for arbitrary structures. In this section, we show that the problem is more complex for datalog programs over data trees.

With respect to (i), we show that the problem of the containment of datalog programs in UCQs over trees (even without data) is undecidable (Corollary 3.2). Indeed, we even

---

1For a definition of query tree automaton, see [15].
prove that the problem of emptiness of datalog programs on
trees without data is undecidable (Proposition 3.1).
With respect to (ii), we show that the problem of con-
tainment of a linear monadic datalog program in a union of
conjunctive queries on data trees is undecidable (Proposi-
tion 3.3).
Lastly, we also consider bounded depth data trees. We
show that the containment of a linear monadic datalog
program in a union of conjunctive queries with data inequal-
ities is undecidable for bounded depth data trees (Propo-
sition 3.5). This is to set limitations to a result of next
section that states that this problem is decidable in absence
of inequalities.

Datalog on trees without data
A datalog program \( P \) over schema \( S \) is empty if for each in-
stance \( I \) over \( S \), \( P(I) \) is empty. Checking datalog program
emptiness is decidable over relational structures [1]. How-
ever, it becomes undecidable if the instances are constrained
to be trees. We believe this result to be known. But since
we could not find a reference for it, we include it here for
completeness.

**Proposition 3.1.** Emptiness of datalog programs over
words or trees with schema \( S \), is undecidable.

**Proof.** We encode a run of a deterministic Turing ma-
chine on empty input by a datalog program over trees as
follows. Let \( \Gamma \) be the tape alphabet, \( Q \) the set of states and
\( \Delta = Q \times \Gamma \rightarrow Q \times \Gamma \cup \{-1, 0, 1\} \) the transition function. A
run \( C_0 \vdash C_1 \vdash \cdots \vdash C_n \) is a sequence of configurations \( C_i \),
where each \( C_i \) is encoded as a word from \( \Gamma^*(Q \times \Gamma)^* \) of
length \( K \) (where \( K \geq \max_i |C_i| \)). The datalog program will
check that an accepting run \( \#C_0 \#C_1 \# \cdots \#C_n \# \) labels some
path leading upwards in the tree.

The program consists of three subprograms, with binary
goal predicates \( R_0, R_\Delta, R_f \). We define \( R_0 \) such that for any
two nodes \( n, n' \), we have \( R_0(n, n') \) iff \( n' \) is an ancestor of \( n \)
and the path from \( n \) to \( n' \) is labeled by \( \#C_0 \# \). The
program \( R_\Delta \) is such that \( R_\Delta(n, n') \) iff \( n' \) is an ancestor of \( n \)
nested to the \( j \)-th position in some configuration \( C \), whereas \( n' \) is the \( j \)-
th position in the configuration \( C' \) with \( C \vdash C' \). Finally, \( R_f \)
is such that \( R_f(n, n') \) iff \( R_\Delta(n, n') \) and \( n' \) has label from
\( \Gamma_F = \Gamma \times F \), with \( F \) the set of accepting states. All three
programs are linear (and binary). Let \( \Gamma_Q = \Gamma \cup \Gamma \times Q \) and
\( S \subseteq (\Gamma_Q)^4 \) be the relation corresponding to the transition
function of the TM: \( (a, \beta, \gamma, \delta) \in S \) iff for some \( C \vdash C' \) and
some position \( 1 \leq j \leq K = |C| = |C'| \), the \( j \)-th symbol in
\( C' \) is \( \delta \), and the \( j-1, j, j+1 \)-th symbols in \( C \) are \( a, \beta, \gamma \),
respectively (for \( i = 0 \) and \( i = K + 1 \) we set the symbol to be \( # \)). Below, we use \( S(x, y, z, w) \) as a shorthand for
\( V_{(a, b, c, d) \in S} \mathbf{a}(x) \land \mathbf{b}(y) \land \mathbf{c}(z) \land \mathbf{d}(w) \) and describe \( R_\Delta \) and
\( R_f \) (\( R_0 \) is similar).

- \( R_\Delta(x, y) \) = \( \Gamma_Q(x), \Gamma_Q(y), \text{child}(x, x_1), \text{child}(y, y_1), \text{child}(z, x), S(x_1, x, z, y), R_0(x_1, y_1) \)
- \( R_\Delta(x, y) = \Gamma_Q(x), \Gamma_Q(y), \text{child}(x, x_1), \text{child}(y, y_1), \text{child}(z, x), S(x_1, x, z, y), R_\Delta(x_1, y_1) \)
- \( R_f(x, y) = R_\Delta(x, y), \Gamma_F(y) \)

Observe that the datalog program in the above proof is
linear and binary (i.e., 2-datalog). Now this implies:

**Corollary 3.2.** Containment of 2-datalog programs in
CQs over words or trees with schema \( S \), is undecidable.

**Proof.** Let \( P \) be a datalog program and let \( Q \) be the conjunc-
tive query \( \exists x \text{child}(x, x) \), then \( Q \) never holds for trees.
Thus over trees, \( P \subseteq Q \) iff \( P \) is empty. \( \square \)

### Linear monadic datalog over data trees

We show the following result:

**Proposition 3.3.** Containment of linear monadic data-
log programs in UCQs over data words or over data trees
with schema \( S^- \), is undecidable.

**Proof.** We first show the result for words. We reduce the
Post Correspondence Problem (PCP) to our problem on
data words. Let \( (u_i, v_i)_{1 \leq i \leq k} \) be a PCP instance over
alphabet \( \Sigma \).

We want to build a linear, monadic datalog program \( P \)
and a UCQ \( Q \) (both Boolean) such that \( P \) is not contained in
\( Q \) iff there exist \( m \geq 1 \) and indices \( i_1, \ldots, i_m \) such that
\( u_1 \cdots u_m = v_1 \cdots v_m \).

We start with an informal description. The linear monadic
program encodes a data word with labels from \( \Sigma \cup \{\#\} \),
where \( \Sigma \) is a (disjoint) copy of \( \Sigma \). The following conditions
should hold:

1. The string projection of this data word belongs to the
set \( \#(\bigcup_{1 \leq i \leq k} u_i \tau)^\# \), where \( \tau \) denotes the copy of
the word \( v \) over \( \bar{\Sigma} \).
2. All positions labelled by \( \Sigma \) (\( \bar{\Sigma} \), respectively) carry dif-
ferent data values.
3. Let \( \rho \) (\( \bar{\rho} \), respectively) denote the sequence of data
values of the subword with labels from \( \Sigma \) (\( \bar{\Sigma} \), re-
spectively). The sequences \( \rho \) and \( \bar{\rho} \) are equal, and any two
positions in \( \rho \) and \( \bar{\rho} \) with equal data value have the
same label (i.e., \( a \) and \( \bar{a} \) for some \( a \in \Sigma \)).

A monadic datalog program \( P \) can check Condition (1),
assuming that we have the special markers \( \#, \$ \) for the first
and last position, respectively (that these markers are used
only at the first and last position, respectively, can be guar-
anteed by the UCQ). More generally, any finite-state auto-

Condition (2) is enforced by the negation of the UCQ \( Q \),
as follows. A UCQ can express the existence of two positions
\( x \neq x \) such that both \( x, y \) have label in \( \Sigma \) (\( \Sigma \), re-
pectively) and \( x \sim y \). Notice that we do not actually need the atomic
predicate \( x \neq y \): due to Condition (1) we can distinguish
several cases, depending on whether \( x, y \) belong to the same
occurrence of some \( u_i \) (\( \bar{\tau} \), respectively), so we can use the
child (successor) predicate, or they are separated by some
symbol from \( \Sigma \) (\( \bar{\Sigma} \), respectively), so we use the desc
(linear order) predicate.

Condition (3) is enforced by the datalog program: First,
\( P \) checks that the first (last, respectively) node with label
from \( \Sigma \) has the same value as the first (last, respectively)
node with label from \( \bar{\Sigma} \), and that the labels are the same.
For identifying these nodes, the special markers \( \#, \$ \) for the
first/last node are used. Then, $P$ checks recursively for every node $x$ in $\rho$: there exists a node $x'$ in $\rho$ such that $x \sim x'$ and $y \sim y'$ hold, where $y$ is the successor of $x$ in $\rho$ and $y'$ is the successor of $x'$ in $\rho$. Notice that $P$ can refer to the successor in $\rho$ (resp. $\bar{\rho}$) by using the child predicate, since the distance between two consecutive nodes labeled by $\Sigma$ ($\bar{\Sigma}$, respectively) is bounded by the maximal length of the words $v_i$ and $u_i$. The requirement for equal labels $a, \bar{a}$ is checked at the same time.

For data trees, the crux is (as in Proposition 3.1) to follow paths leading upwards in the tree. One needs to pay attention in particular to Condition (3) above. □

It is interesting to note that Proposition 3.3 remains true even for programs using desc but not child. It remains open whether using child, but not desc, remains undecidable.

**Proposition 3.4.** Containment of linear monadic datalog programs in UCQs over data words or trees (with schema $S^{-\sim}$) is undecidable, even for programs without the child relation.

**Bounded depth and inequalities**

The inclusion problem for conjunctive queries over data trees where queries can use data inequalities, is known to be undecidable [5]. We will prove in the next section that containment of a linear monadic datalog program in a UCQ is decidable for bounded depth data trees, in absence of node and data inequalities. This is in the spirit of [12], which showed that satisfiability of certain queries restricted to trees of bounded depth is decidable, while being undecidable for unbounded depth. We next show that the decidability result in the next section cannot be extended to datalog and UCQs using inequalities.

**Proposition 3.5.** Containment of linear, monadic datalog programs in UCQs over schema $S^{-\sim}$, is undecidable for data trees of depth 2.

**Proof.** We simulate the proof of Proposition 3.3 on data trees of depth 2 as follows. The basic idea is to encode a list $L$ on (some of) the nodes at depth 1. We describe first how trees satisfying $P \cap \neg Q$ should look like.

1. Nodes at depth 1 have labels from $\Sigma \cup \bar{\Sigma}$, and nodes at depth 2 have label $L$ or $R$.
2. If an $L$-node and an $R$-node have the same parent, then they have different data.
3. If two $L$-nodes have the same parent, then they have equal data, too. Same for $R$-nodes.
4. If two nodes $x,y$ at depth 1 are such that their $L$-children have equal data, then their $R$-children also have equal data (if they exist), and both the labels and the data of $x,y$ are identical. Same for $L,R$ interchanged.
5. There are two distinguished nodes at depth 1, one denoted as “start”, the other as “end”.
6. Node “end” can be reached from “start” by iterating the following successor relation: $y$ is the successor of $x$ if the data of the $L$-child of $y$ is equal to the data of the $R$-child of $x$. Notice that there might be several nodes $y$ that satisfy this condition, but they all agree in their label and data (Condition (4)). So the path from “start” to “end” is unique (if it exists) if we consider the label/data of the nodes on the path.

Note that the datalog-$^\sim$ program $P$ can enforce Condition (6) (and the existence of “start”, “end”), whereas the remaining conditions are expressed by the negation of $Q$.

We show how to mimic the proof of Proposition 3.3 on the list $L$ from “start” to “end”.

First, notice that the program $P$ can verify the first and last condition in the proof of Proposition 3.3, by restricting the checks to nodes in $L$. It suffices to start the checks at node “start” - the unicity of the path from “start” to “end” ensures that we are considering only nodes in the list $L$.

The condition ensured by the UCQ in the proof of Proposition 3.3 is that there exist two nodes $x \neq y$ both with label from $\Sigma$ (or $\bar{\Sigma}$) such that $x \sim y$. We can express $x \neq y$ by saying that the $L$-child of $x$, say $x'$, and the $L$-child of $y$, say $y'$, are such that $x' \sim y'$. □

**4. CONTAINMENT PROBLEM: DECIDABILITY**

As shown in the previous section, the static analysis of datalog programs over trees rapidly leads to undecidability results. In this section, we show two positive results about query containment for data trees, both for trees of bounded depth. A third positive result for the emptiness problem over unbounded depth trees is presented in the next section. As already mentioned, the undecidability results in Section 2 rely in an essential way on the presence of data values. (Recall the decidability results mentioned in Section 2.) Interestingly, we next show that one regains decidability in the presence of data values by imposing that the depth is bounded. We next consider inclusion of datalog in UCQs, then monadic datalog.

**Bounded depth, datalog and UCQs**

We show next that surprisingly, containment of datalog programs into UCQs becomes decidable over bounded depth data trees. Note that the restriction to bounded depth trees is reasonable, since in practice, XML trees are quite often of very small depth.

**Theorem 4.1.** Let $k > 0$ be a fixed integer. The problem of containment of datalog programs in UCQs with schema $S^{-\sim}$ over data trees of depth less than $k$, is decidable. The complexity is in 3EXPTIME and 2EXPTIME-hard.

**Proof.** Let $P$ be a datalog program and $Q$ be a UCQ with the schema $S^{-\sim}$. We assume without loss of generality that both $P,Q$ are Boolean. The proof goes as follows. We define a new relational schema $\bar{S}$ and transform $P$ and $Q$ into a datalog program $P'$ and a UCQ $Q'$ over $\bar{S}$ (both Boolean) such that:

$$P \subseteq Q$$

over data trees of depth at most $k$ iff $P' \subseteq Q'$.

Then using Theorem 2.1, this will show decidability. The key point in the translation is as follows. Each node $n$ at depth $j \in \{0, \ldots, k\}$ in the data tree $t$ will be identified by a relational tuple $(n_0, v_0, \ldots, n_j, v_j)$. The $n$-components in the tuples are needed for translating the predicate $\sim$. Different tuples represent different tree nodes, and the node
encoded by \((n_0, v_0, \ldots, n_j, v_j)\) is a child of the node encoded by \((n_0, v_0, \ldots, n_{j-1}, v_{j-1})\). Moreover, two tuples \((n_0, v_0, \ldots, n_i, v_i)\) and \((n_0, v_0, \ldots, n, w_1, \ldots)\) with \(v_i \neq w_i\) represent different tree nodes. We will need to assert some more things in order to ensure that a relational structure can be indeed decoded into a data tree.

We will do the transformation for each rule \(r\) of \(P\) separately. We show first that we may assume that the predicate \(desc\) does not occur in the body of \(r\), at the cost of increasing the number of rules by \(k^d\), where \(d\) is the maximal number of \(desc\) atoms in the body of a rule. We may do this by replacing every atom \(desc(x, y)\) by some conjunction \(child(x, z_1), child(z_1, z_2), \ldots, child(z_i, y), 0 \leq i < k - 1\), where \(z_1, \ldots, z_i\) are new variables (by convention, for \(i = 0\) the sequence equals \(child(x, y)\)).

Without loss of generality, we also assume that (***) no rule contains atoms \(child(x, z)\) and \(child(y, z)\) with \(x \neq y\), since in this case, \(x, y\) denote the same node in the data tree and \(y\) can be replaced by \(x\) or vice versa. In the following, we show how to transform every rule \(r\) over \(S^r\) into a set of datalog rules over the relational schema \(S' = \{\alpha_j | \alpha \in \Sigma, 0 \leq j \leq k\}\). First we fix a new variable name \(u\), which will stand for the root and will be used in all transformations of rules.

Let \(Var(r)\) be the set of variables in the body of a rule \(r\). We assign to each variable \(x\) in \(Var(r)\) a depth: let \(D : Var(r) \rightarrow \{0, \ldots, k\}\). This assignment is done consistently with the \(child(x, y)\) atoms in \(r\), so that \(D(y) = D(x) + 1\) must hold in that case. For each variable \(x\) in \(r\) such that there is no atom \(child(z, x)\) in \(r\) we also add to \(r\) the sequence of atoms \(child(u, x_1), child(x_1, x_2), \ldots, child(x_{D(u)-1}, x)\), where \(x_1, \ldots\) fresh variables (we might have \(x_i = y_i\) for some \(x \neq y\), but we should respect the constraint (***) above). We call the modified rule \(r(D)\).

Let us consider a rule \(r(D)\) modified as above and choose a labeling \(l(x) \in \Sigma\) for the variables in \(r(D)\). This choice should be consistent with the atoms of the form \(\alpha(x)\) in \(r(D)\). Note that there are at most \((k+1)|\Sigma|^{\mid Var(r)\mid}\) many pairs \((r(D), \ell)\).

For each pair \((r(D), \ell)\) we define now a rule \(r'\) over \(S'\) as follows:

1. For every variable \(z\) in \(r(D)\) add a new variable, say \(v_z\). The variable \(v_z\) associated with the “root” \(u\) should be the same for all transformations.

Suppose that \(u, z_1, \ldots, z_i = z\) is the “path” from the “root” \(u\) to variable \(z\) in \(r(D)\), with \(D(z) = j\). Set \(\pi_j\) to be the sequence of variable-pairs \((u, v_u), (z_1, v_{z_1}), \ldots, (z_j, v_{z_j})\).

2. For every intensional predicate \(R(x_1, \ldots, x_l)\) in \(r(D)\) there is a (new) intensional predicate \(R'(\pi_{x_1}, \ldots, \pi_{x_l})\) in \(r'\), of arity \(\sum_{i=1}^l D(x_i)\).

3. For each variable \(x\) in \(r(D)\) with \(D(x) = j\) and \(\ell(x) = \alpha\), we have an atom \(\alpha_j(\pi_x)\) in \(r'\).

4. For every intensional predicate \(x \sim y\) in \(r(D)\) we have the atom \(v_x = v_y\) in \(r'\).

The datalog program \(P'\) over \(S'\) consists of all rules \(r'\) obtained from some pair \((r(D), \ell)\) as above. The size of \(P'\) is at most exponential in the size of \(P\).

Similar transformations can be applied to each disjunct in the UCQ \(Q\) and we get an UCQ \(Q'\) over \(S'\).

There remains one final issue: by choosing the mapping \(\ell\) we ensure that every node in the tree decoded from some instance over \(S'\) has a label from \(\Sigma\). However we also need to guarantee that the label is unique. To ensure this, for every \(i : 0 \leq i \leq k\) and \(e, \beta \in \Sigma\) with \(e \neq \beta\), we add the formula \(\exists x_0, v_0, \ldots, v_i, v_i(\alpha_i(x_0, v_0, \ldots, x_i, v_i) \land \beta_i(x_0, v_0, \ldots, x_i, v_i))\) as a disjunct of \(Q'\).

Finally we show that \(P \subseteq Q\) over data trees of depth at most \(k\) if \(P' \subseteq Q'\).

Suppose first that there is some data tree \(t\) (of depth at most \(k\)) such that \(P(I)\) holds but \(Q(I)\) does not, for some \(I\) representing \(t\) over \(S^r\). Let \(I'\) be some representation of \(t\) over \(S^{r'}\), according to our encoding of nodes by sequences of nodes/values. It is easy to check that \(P'(I')\) holds, but \(Q'(I')\) does not.

Conversely, assume that \(P'\) holds but \(Q'\) does not, and let \(I'\) be an instance over \(S'\) witnessing this. Consider a proof tree showing that \(P'(I')\) holds, and let \(I''\) be the instance obtained from \(I'\) by keeping each fact occurring in this proof tree. Observe that \(I''\) is the representation of a (data) tree \(t\) of depth at most \(k\), since there is a unique root node (corresponding to the variable pair \(u, v_u\)), and each node (corresponding to a distinct tuple) has a unique label from \(\Sigma\) - otherwise \(Q'(I'')\) would hold. Moreover, it is easy to check that \(P\) holds on \(t\), but \(Q\) does not.

Complexity upper bound. First there is an exponential blow-up in the translation from \(P\) to \(P'\) and from \(Q\) to \(Q'\). Due to the fact that the containment of a datalog program in a UCQ over arbitrary relational structures is in 2EXPTIME, we conclude that the containment of datalog programs in UCQs over the schema \(S^r\) is in 3EXPTIME.

Hardness. 2EXPTIME hardness is proved by a reduction from the problem of containment of a datalog program \(P\) in a UCQ \(Q\) over arbitrary relational structures. Consider a relational schema \(\mathcal{R}\). An instance of \(\mathcal{R}\) is represented as a tree with one subtree labelled \(p\) of the root for each relation \(p \in \mathcal{R}\). The \(p\)-node has a child labeled \(p_{up}\) for each tuple \(p\) with children labelled \(p_1, \ldots, p_i\) if \(p\) has arity \(i\). In each rule \(r\) of \(P\), an atom \(p(x_1, \ldots, x_l)\) is translated into the conjunction of the following atoms, \(\text{root}(y), child(y, z), \text{p}(z), child(z, x), p_{up}(x), child(x, x'), \text{p}(x'), child(x', x), \ldots, child(x, x'), p(x'), child(x', x), \) where \(x, y, z, x', \ldots, x'\) are new variables. In addition, an atom \(x_1 = x_2\) is replaced by \(x_1 \sim x_2\). Let \(P', Q'\) denote the resulting datalog program and the UCQ. Then one can show that \(P \subseteq Q\) over the instances of \(\mathcal{R}\) iff \(P' \subseteq Q'\) over bounded depth data trees.

As a corollary we have:

**Corollary 4.2.** Let \(k > 0\) be fixed. The emptiness problem for datalog programs over data trees of depth less than \(k\) over the schema \(S^r\), is decidable.

**Bounded depth and monadic datalog**

Our result in this section shows that the containment problem for monadic datalog programs over bounded depth data trees is also decidable. For comparison, containment of
monadic datalog programs over relational structures is \textsc{2Exptime}-complete [10, 3]. We do not have a precise complexity bound for data trees, as explained below.

We use the following result\(^2\), which is a specialization of Theorem 5.5 of [11] to datalog programs. Note that the upper bound in the corollary below is (at least) non-elementary, since [11] talks about inclusion of hypergraph grammars in monadic second-order logic. We conjecture that the complexity is lower for data trees.

\textbf{COROLLARY 4.3.} The problem of containment of datalog programs in monadic datalog programs over relational schemes is decidable.

\textbf{Proof.} It is known that each Boolean monadic datalog program over a relational schema can be rewritten into an MSO formula without free variables over the same schema. In addition, properties expressed by Boolean monadic datalog programs are closed under homomorphisms, as a result of the lack of negation symbols in datalog programs. Then the result follows from Theorem 5.5 in [11]. \(\square\)

Now we have the following result.

\textbf{THEOREM 4.4.} The problem of containment of datalog programs in monadic datalog programs with the schema \(S^*\) over bounded depth data trees is decidable.

\textbf{Proof.} We reduce the problem to the problem of containment of datalog programs in monadic datalog programs over relational structures. Let \(P\) and \(Q\) be two datalog programs over \(S^*\), with \(Q\) monadic. We assume that \(P\) does not use node equalities (they can be replaced by syntactically replacing variable names). We translate \(P\) to a relational datalog program \(P'\) using the construction in the proof of Theorem 4.1. Note that \(P'\) does not use variable equalities either. We would like to proceed similarly with \(Q\), to obtain \(Q'\). However doing it naively would lead to a non-monadic program, since a variable in \(Q\) is simulated by a tuple of variables in \(Q'\). Nevertheless, it turns out that it suffices to use one variable to represent the tuple of variables in \(Q'\). The main idea is that \(Q'\) does not need to enforce the tree structure anymore, since \(P'\) does. So let \(Q'\) be the non-monadic datalog program obtained from \(Q\) as in the proof of Theorem 4.1. We define \(Q''\) as the monadic datalog program obtained from \(Q'\) by replacing each intensional atom \(R'(x_0, v_0, \ldots, x_i, v_i)\) by \(R(x_i)\). If \(R'(x_0, v_0, \ldots, x_{i-1}, v_{i-1}, v_i)\) was the head of a rule, this means of course that \(x_0, v_0, \ldots, x_{i-1}, v_{i-1}, v_i\) will be quantified existentially in the body of the rule.

In the following, we will show that

\[ P \subseteq Q \text{ over data trees of depth at most } k \text{ iff } P' \subseteq Q''. \]  

\((*)\)

To prove \((*)\), we need to make sure that in a relational instance \(a_{2i}\) properly identifies the node of the tree that we previously identified by the tuple \((a_{0}, a_{1}, \ldots, a_{2i-1}, a_{2i+1})\). For that, consider the following property for the instances \(I\) over \(S'\). Assume that \((a_{0}, a_{1}, \ldots, a_{2i-1}, a_{2i+1})\) and \((b_{0}, b_{1}, \ldots, b_{2i}, b_{2i+1})\) are two tuples occurring in the interpretations of some extensional predicate \(\alpha_{i}\) in \(I\). Then the following two conditions hold:

1. if \(a_{2i} = b_{2j}\), then for each \(j' < 2j\), \(a_{j'} = b_{j'}\), and \(a_{2j+1} = b_{2j+1}\).
2. for each \(j \neq j'\), \(a_{2j} \neq a_{2j'}\).

First suppose \(P' \not\subseteq Q''\). Let \(I\) be an instance satisfying \(P'\) but not \(Q''\). Now, consider the proof tree that demonstrates \(P'\) using \(I\). We can assume without loss of generality that \(I\) only contains facts occurring in that proof tree. Similar to the argument in the proof of Theorem 4.1, we also observe that \(I\) is the representation of a data tree \(t\) of depth at most \(k\). The instance \(I\) may not satisfy conditions (1) and (2) above. But for every tuple \((n_0, v_0, \ldots, n_i, v_i)\) in \(I\), we can choose a tuple of new elements \((n'_0, \ldots, n'_i)\) and replace the tuple \((n_0, v_0, \ldots, n_i, v_i)\) by \((n'_0, v_0, \ldots, n'_i, v_i)\), so that conditions (1) and (2) above are satisfied. We do this by preserving common prefixes: if \((n_0, v_0, \ldots, n_i, v_i)\) is a common prefix of two tuples, then we use the same new elements \((n'_0, \ldots, n'_i)\) in both tuples. Let \(I'\) be the resulting instance.

It is easy to observe that there is a (non-injective) mapping \(\varphi\) from the domain of \(I'\) to that of \(I\) such that \(\varphi\) is consistent with the interpretations of the relations \(\alpha_{i}\) in \(I'\) and \(I\), more precisely, if \(\alpha_{i}(n_0, v_0, \ldots, n_i, v_i)\) holds in \(I'\), then \(\alpha_{i}(\varphi(n_0), \varphi(v_0), \ldots, \varphi(n_i), \varphi(v_i))\) holds in \(I\). Clearly, \(I'\) satisfies \(P'\), because I already did and because \(P'\) does not use equalities. We show by contradiction that \(I'\) does not satisfy \(Q''\). Because \(Q''\) contains no negations, if \(I'\) would satisfy \(Q''\) then \(I'\) would satisfy \(Q''\) as well, a contradiction. Therefore, we get an instance \(I'\) which satisfies \(P'\) but not \(Q''\), and that satisfies the condition (1) and (2) above. Since \(I'\) is also a representation of \(t\), it follows that \(t\) satisfies \(P\). Since \(I'\) satisfies conditions (1) and (2), we also infer that \(t\) does not satisfy \(Q\). We conclude that \(P \not\subseteq Q\).

Now suppose \(P \not\subseteq Q\). Let \(t\) be a bounded depth data tree satisfying \(P\) but not \(Q\), and \(I\) be an encoding of \(t\), satisfying conditions (1) and (2). Then \(P'\) and \(Q''\) on \(I\) correctly simulate \(P\) and \(Q\) on \(t\), respectively, thus \(I\) satisfies \(P'\) but not \(Q''\). Hence \(P' \not\subseteq Q''\). \(\square\)

5. \textbf{EMPTINESS PROBLEM AND PATTERN AUTOMATA}

In this section, we define pattern automata that are essentially equivalent to linear datalog programs over data trees. Using these automata, we show that over data trees, the emptiness problem for datalog\(^-\cdot\) programs is decidable.

We first describe pattern automata informally. They use data tree patterns in the form of state invariants and transition constraints. Transitions are guarded by constraints that relate variables of the source and the target state.

Data tree patterns (patterns for short) are conjunctive queries over the schema \(S\)\(^-\cdot\), or equivalently, formulas of the form \(\exists x_1 \cdots \exists x_k \varphi\) with \(\varphi\) a conjunction of positive atomic formulas over \(S\)\(^-\cdot\). Free variables of patterns are defined as usual. For a pattern \(P\), let Var\((P)\) (resp. Free\((P)\)) denote the set of variables (resp. free variables) occurring in \(P\). Matchings of patterns into data trees have the standard (non-injective) meaning. Let also \(P\) denote the collection of all patterns. It is easy to see that the satisfiability problem of patterns can be solved in NP.

For notational convenience, in this section, we will use the schema \(S\)\(^-\cdot\)\(^*\) which is the extension of the schema \(S\)\(^-\cdot\) with child\(^{-1}\) and desc\(^{-1}\), the inverse relation of child and desc. The formula child\(^{-1}\)(\(x, y\)) and desc\(^{-1}\)(\(x, y\)) can be
thought as respectively the formula $child(y, x)$ and $desc(y, x)$ within the schema $S^{\sim, -1}$.

A constraint $\psi$ is a conjunction of atomic formulas over $S^{\sim, -1} \Sigma$ and variables $x, y, \ldots$ and $x', y', \ldots$. Let $\mathcal{E}$ denote the collection of all constraints. For $\psi \in \mathcal{E}$, let $\operatorname{Var}(\psi)$ denote the set of variables occurring in $\psi$.

**Definition 5.1.** A pattern automaton (PA for short) $A$ is a tuple $(\mathcal{Q}, I, F, \gamma, \Delta)$ such that

- $\mathcal{Q}$ is a finite set of states, $I \subseteq \mathcal{Q}$ is the set of initial states, $F \subseteq \mathcal{Q}$ is the set of final states,
- $\gamma : \mathcal{Q} \rightarrow P$ assigns to each state in $\mathcal{Q}$ a pattern (state invariant),
- $\Delta \subseteq \mathcal{Q} \times \mathcal{E} \times \mathcal{Q}$ is a set of guarded transitions: for each $(\mathcal{q}, \psi, \mathcal{q'}) \in \Delta$, the constraint $\psi$ is such that each atom is of the form $\alpha(x, x')$, where $\alpha \in S^{\sim, -1} \setminus \Sigma$, $x \in \operatorname{Free}(\gamma(q))$, and $x' \in \operatorname{Free}(\gamma(q'))$. Notice that the first variable in $\psi$ refers to $q$, whereas the second one refers to $q'$.

We assume for the remaining of the section that all state invariants and transition constraints of PA are satisfiable$^3$.

A run of $A$ over a data tree $t$ is a sequence 

$$(q_0, \pi_0)(q_0, \psi_1, q_1)(q_1, \pi_1) \cdots (q_{k-1}, \pi_{k-1})(q_{k-1}, \psi_k, q_k)(q_k, \pi_k)$$

such that for every $i$,

- $(q_i, \psi_{i+1}, q_{i+1}) \in \Delta$,
- $\pi_i$ is a matching of $\gamma(q_i)$ into $t$,
- let $\psi_{i+1} = \bigcup_j \alpha_j(x_j, x'_j)$, where for every $j$: $\alpha_j \in S^{\sim, -1} \setminus \Sigma, x_j \in \operatorname{Free}(\gamma(q_i))$, $x'_j \in \operatorname{Free}(\gamma(q_{i+1}))$; moreover, $\alpha_j(\pi_i(x_j), \pi_{i+1}(x'_j))$ holds in $t$.

A run $(q_0, \pi_0) \cdots (q_k, \pi_k)$ on $t$ as above is accepting if $q_0 \in I$ and $q_k \in F$. The PA $A$ is said to be non-empty if it has some accepting run. Let $A = (\mathcal{Q}, I, F, \gamma, \Delta)$ be a PA. The arity of a transition $(\mathcal{q}, \psi, \mathcal{q'}) \in \Delta$ is defined as

$$\max(|\operatorname{Free}(\gamma(q)) \cap \operatorname{Var}(\psi)|, |\operatorname{Free}(\gamma(q')) \cap \operatorname{Var}(\psi)|).$$

The arity of a PA is the maximum of the arities of its transitions.

A PA of arity one is called unary. So, if $A$ is unary then each transition $(\mathcal{q}, \psi, \mathcal{q'})$ of $A$ can be written as $\psi(x, x') = \bigcup_x \alpha(x, x')$ for some $x \in \operatorname{Free}(\gamma(q))$, $x' \in \operatorname{Free}(\gamma(q'))$ and each $\alpha$ is from $S^{\sim, -1} \setminus \Sigma$.

**Example 5.2** A pattern automaton $A = (\mathcal{Q}, I, F, \gamma, \Delta)$ is illustrated in Figure 1, where

- $\mathcal{Q} = \{q_0, q_1, q_2\}$, $I = \{q_0\}$, $F = \{q_2\}$,
- $\gamma$ is defined as follows,

$\gamma(q_0) = \exists x_2 \exists x_3 (a(x_1) \land b(x_2) \land c(x_3) \land child(x_1, x_2) \land desc(x_1, x_3) \land x_2 \sim x_3)$,

$\gamma(q_1) = \exists x_2 (b(x_1) \land b(x_2) \land c(x_0) \land child(x_4, x_5) \land desc(x_4, x_6) \land x_5 \sim x_6)$.

$\Delta = \{(q_0, x_2 = x_4, q_1), (q_1, x_5 = x_7, q_2), (q_1, x_5 = x_7, q_2)\}$.

Intuitively, $A$ starts from an $a$-node, looks downwards for a sequence of $b$-nodes $v$, each of them having another $c$-node $v'$ such that $v'$ is a descendant of the parent of $v$, and $v'$ has the same data value as $v$; finally, $A$ stops at a $d$-node $w$ such that there is a $c$-node $w'$ which is a descendant of the parent of $w$, and has a different data value from $w$.

![Figure 1: Example for pattern automata](image-url)

The following theorem shows that pattern automata are equivalent to (Boolean) linear datalog$^{\sim, -1}$ programs.

**Theorem 5.3** The following result holds.

1. For any Boolean linear datalog$^{\sim, -1}$ program $P$, a PA $A$ of the same arity as $P$ can be constructed in linear time such that for every data tree $t$, $P$ evaluates to true over $t$ if and only if $t$ is accepted by $A$.

2. For any PA $A$, a Boolean linear datalog$^{\sim, -1}$ program $P$ of the same arity as $A$ can be constructed in linear time, such that for every data tree $t$, $t$ is accepted by $A$ if and only if $P$ evaluates to true over $t$.

**Proof.** (Sketch) A PA can simulate a datalog$^{\sim, -1}$ program with states corresponding to rules, state invariants corresponding to the body of the corresponding rule, and transition constraints relating the variables of the intensional predicate of the body with the one of the head. Conversely, a datalog$^{\sim, -1}$ program simulates a PA by using one intensional predicate for each transition. $\blacksquare$

**Example 5.4** We first illustrate how to transform the PA $A$ in Example 5.2 into a datalog$^{\sim, -1}$ program $P_A$. Then we show how to translate $P_A$ back into a (different) PA $B$.

Using one intensional predicate for each transition, say $R_1, R_2, R_3$ for respectively the transition $(q_0, x_2 = x_4, q_1), (q_1, x_5 = x_7, q_2), (q_1, x_5 = x_7, q_2)$, the datalog$^{\sim, -1}$ program $P_A$ for $A$ in Example 5.2 is constructed as follows,

- $r_1 : R_0() \triangleright \gamma(q_0), R_1(x_2)$,
- $r_2 : R_1(x_2) \triangleright x_2 = x_4, \gamma(q_1), R_2(x_3)$,
- $r_3 : R_1(x_2) \triangleright x_2 = x_4, \gamma(q_1), R_3(x_5)$,
- $r_4 : R_2(x_3) \triangleright x_3 = x_5, \gamma(q_1), R_2(x_5)$,
- $r_5 : R_2(x_3) \triangleright x_3 = x_5, y'(q_1), R_3(x_5)$,
- $r_6 : R_3(x_5) \triangleright x_5 = x_7, \gamma(q_2)$.

---

$^3$This check can be seen as a preprocessing in EXPTIME and will not affect the 2EXPTIME upper bound in Theorem 5.5.
where $x_i'$s are new variables and $\gamma'(q_1)$ is obtained from $\gamma(q_1)$ by replacing each variable $x_i$ with $x_i'$.

Note here we abuse the notations a bit: For instance, by $\gamma(q_0)$, we mean the tuple of atomic formulas occurring in $\gamma(q_0)$. Similarly for $\gamma(q_1), \gamma(q_2), \gamma'(q_1)$.

With one state per rule, we can translate $PA$ back into the PA $B = \{(r_1, \ldots, r_n), \{r_1, \{r_6, \gamma_1, \Delta_1\}\} \}$ illustrated in Figure 2, where for every $i : 1 \leq i \leq 6$, $\gamma_1(r_i)$ is an existential quantification of the conjunction of the extensional atoms in the body of the rule $r_i$, e.g. $\gamma_1(r_2) =$ $\exists x_4 \exists x_6 (x_2 = x_4 \land \gamma(q_1))$, with variables $x_2$ and $x_5$ free.

Figure 2: Transformation from datalog$^{\neg,\neg}$ to PA

From Theorem 5.3 and Proposition 3.1, we know that the nonemptiness problem of PA of arity two is already undecidable over data trees (actually even over trees). On the other hand, in the following, we will prove that the nonemptiness problem for unary PA is decidable over data trees.

**Theorem 5.5.** The nonemptiness problem for unary PA over data trees is in $2\text{Exptime}$ and $\text{Exptime}$-hard.

Using the previous theorem, it follows that:

**Corollary 5.6.** The emptiness problem for linear monadic datalog$^{\neg,\neg}$ programs over data trees is in $2\text{Exptime}$ and $\text{Exptime}$-hard.

Before giving the proof of the theorem, we need some additional notations.

Let $P$ be a pattern and $x$ a variable in $P$. Then the set of all variables $y$ in $P$ (in particular, $x$ itself) such that there is a sequence of variables $z_1, \ldots, z_k$ satisfying that $z_1 = x$, $z_k = y$ and for every $i : 1 \leq i \leq k$, $\text{child}(z_i, z_{i+1})$ or $\text{child}(z_{i+1}, z_i)$ or $z_i = z_{i+1}$, is called a component of $P$. A local pattern is a pattern with a unique component, which is the set of all its variables. Then a non-local pattern has at least two distinct components.

**Definition 5.7.** Let $A = (Q, I, F, \gamma, \Delta)$ be a unary PA. A run $\rho$ of $A$ is said to be local if the following two conditions are satisfied:

- for every $(q, \psi, q')$ in $\rho$, the constraint $\psi(x, y)$ has an atomic formula of the form $\text{child}(x, y)$, or $x = y$.
- for every two consecutive transitions $(q, \psi, q'), (q', \psi', q'')$ in $\rho$, if $\psi(x, y)$ and $\psi'(x', y')$ (where $x \in \text{Free}(\gamma(q))$, $y \in \text{Free}(\gamma(q'))$, and $x' \in \text{Free}(\gamma(q''))$, then $x', y$ belong to the same component of $\gamma(q'')$. Notice that in this case, every matching of $\gamma(q')$ maps $x'$, $y$ to nodes that are at bounded distance in the tree.

A run $\rho$ of $A$ is said to be global if for every $(q, \psi, q')$ in $\rho$, the constraint $\psi(x, y)$ is such that every conjunct is of the form $\alpha(x, y)$ with $\alpha \in \{\text{desc}, \text{desc}^{-1}, \sim, \propto, \not\}$.

A two-way nondeterministic tree walking automaton over ranked trees (TWA for short) is a finite state automaton with transitions defined by the rules of the following form: “if the current state is $q$ and the reading head is at some node $n$ labeled by $\sigma$, then the state changes to $q'$, and the reading head goes to the parent of $n$ or goes to the $i$-th child of $n$ (where $i \in \mathbb{N}$)”. Such an automaton starts at the root and accepts it if it reaches a final state.

We are ready to present the proof of Theorem 5.5 and we first focus on the upper bound.

**Upper bound**

The proof schema of the upper bound goes as follows.

1. First we show how to decide in polynomial time whether a unary PA has a global accepting run.

2. Second we show how to decide in doubly exponential time whether a unary PA has some local accepting run. Here we use tree-walking automata.

3. Finally, we combine the previous proof ideas to show the general case.

For the proposition below, recall that we assume that all invariants and constraints in our PA are satisfiable.

**Global runs for unary PA**

**Proposition 5.8.** The following question is in $\text{PTIME}$. Given a unary PA and two states $q, q'$, does a data tree exist with a global run from $q$ to $q'$?

**Proof.** We prove the result by showing the following claim.

**Claim.** There exists a data tree with a global run from $q$ to $q'$ iff state $q'$ can be reached from $q$ in the transition graph of $A$ by global transitions, only.

The “only if” direction is immediate. For the “if” direction let us consider a path $(q_0, \psi_1, q_1)(q_1, \psi_2, q_2), \ldots, (q_{k-1}, \psi_k, q_k)$ in the transition graph of $A$, such that $q = q_0$, $q_k = q'$, and all transitions $(q_i, \psi_i, q_{i+1})$ are global. Recall that we assume that all state invariants (and constraints) are satisfiable. So for each $i$, let $s_i$ be a data tree such that there is a matching from $\gamma(q_i)$ to $s_i$. We show how to construct a data tree $t$, together with a run

$$(q_0, \pi_0)(q_0, \psi_1, q_1)(q_1, \pi_1) \cdots (q_{k-1}, \psi_k, q_k)(q_k, \pi_k)$$

of $A$ on $t$.

Suppose that $t'$ is already defined, together with a run $(q_0, \pi_0)(q_0, \psi_1, q_1)(q_1, \pi_1) \cdots (q_i, \pi_i)$ on $t'$. The construction procedure goes as follows.

- If $\psi_{i+1}$ requires desc$(x, y)$ (with $x$ in $\gamma(q_i)$, $y$ in $\gamma(q_{i+1})$), then let $\rho$ be a path in $t'$ to which $\pi_i(x)$ belongs and $n$ be the leaf of $t'$ in $\rho$. We extend $t'$ by attaching $s_{i+1}'$ as a subtree of $n$, where $s_{i+1}'$ is obtained from $s_{i+1}$ by renaming possibly data values to account for data constraints in $\psi_{i+1}$.
If $\psi_{i+1}$ requires $\text{desc}^{-1}(x,y)$ (with $x$ in $\gamma(q_i)$, $y$ in \(\gamma(q_{i+1})\)), then let $\pi$ be a matching of $\gamma(q_{i+1})$ to $s_{i+1}$, $\rho$ be a path in $s_{i+1}$ to which $\pi(y)$ belongs to, and $n$ be the leaf of $s_{i+1}$ in $\rho$. We extend $t'$ by attaching $t''$ as a subtre of $n$ in $s_{i+1}$, where $s'_{i+1}$ is obtained from $s_{i+1}$ by renaming possibly data values to account for data constraints in $\psi_{i+1}$.

- Otherwise, we extend $t'$ by choosing an arbitrary node $n$ in $t'$ and attaching $s'_{i+1}$ as a subtree of $n$, where $s'_{i+1}$ is obtained from $s_{i+1}$ just as before.

\[ \square \]

Step 2 of the upper bound proof is more complicated. We start with the slightly simpler case where all state invariants $\gamma(q)$ of a unary PA $A$ are local patterns.

Suppose $A = (Q, I, F, \gamma, \Delta)$ is a unary PA such that all its state invariants are local patterns. Because we are concerned with the existence of local accepting runs, we assume that each transition $(q, \psi, q')$ in $A$ is such that the constraint $\psi(x, y)$ is local, i.e. $\psi(x, y)$ contains an atomic formula $\text{child}(x, y)$ or $\text{child}^{-1}(x, y)$ or $x = y$. Let $\mathcal{P}_A$ be the collection of all local patterns occurring in $A$.

In the following, when we mention accepting runs of $A$, we always mean local accepting runs.

**Proposition 5.9.** Let $A$ be a unary PA such that all its state invariants are local patterns and for each transition $(q, \psi, q')$ of $A$, the constraint $\psi(x, y)$ is local. Then $A$ can be transformed into an equivalent unary PA $A_1$ in linear time such that all the state invariants of $A_1$ are still local patterns, and for each transition $(q, \psi, q')$ of $A_1$, the constraint $\psi(x, y)$ is equal to $x = y$.

By Proposition 5.9 we assume in the following that:

For every transition $(q, \psi, q')$ in $A$, $\psi$ is equal to $x = y$ for some variables $x, y$.

The algorithm to decide the existence of local accepting runs for $A$ (for local invariants) goes as follows.

1. At first, a concept of “backbones of accepting runs” is defined and it is shown that the nonemptiness of $A$ can be reduced to the nonemptiness of a PA $A'$ to check the existence of “accepting” backbones. Moreover, the size of $A'$ is polynomial over the size of $A$.

2. We show that if $A'$ is nonempty, then it accepts a data tree where every node have exponentially many children (with respect to the size of $A'$).

3. In addition, we prove that if $A'$ is nonempty, then $A'$ accepts a data tree containing only exponentially many data values.

4. Finally $A'$ can be seen as a TWA $B$ over ranked (non-data) trees with exponentially many states.

Because it is known that the nonemptiness of a TWA over ranked trees can be solved in exponential time, it follows that the existence of local accepting runs of a unary PA where all the state invariants are local patterns, can be solved in doubly exponential time.

Before presenting the detailed proof, we introduce some vocabulary. Let $t$ be a tree or data tree. We use paths and walks to distinguish between directed paths and undirected paths in $t$. A path from $n$ to $n'$ in $t$ is a sequence of nodes $n_1, \ldots, n_k$ in $t$ such that $n_1 = n, n_k = n'$, and for every $i : 1 \leq i < k, \text{child}(n_i, n_{i+1})$ holds in $t$. Note that the sequence $n_1, \ldots, n_k$ in a path of $t$ must be non-repeating. On the other hand, a walk between $n$ and $n'$ in $t$ is a sequence of vertices $n_1, \ldots, n_k$ such that $n_1 = n, n_k = n'$, and for every $i : 1 \leq i < k, \text{child}(n_i, n_{i+1})$ holds in $t$. A walk $n_1 \ldots n_k$ is simple if $n_1 \ldots n_k$ is non-repeating. It is easy to observe that for every two distinct nodes in trees or data trees, there is a unique simple walk between them.

**Backbones of accepting runs**

Let $\rho = (q_0, \pi_0)(q_0, \psi_1, q_1)(q_1, \pi_1)\ldots(q_{k-1}, \psi_k, q_k)(q_k, \pi_k)$ be an accepting run of $A$ over a data tree $t$.

Let $x_0, x_1, x_2, \ldots, x_{k-1}, x_k$ be a list of variables such that $\psi_1 = (x_0 = x_1), \psi_k = (x_{k-1} = x_k)$, and for every $1 \leq i < k, \psi_i = (x_{i-1} = x_i)$. In addition, for every $1 \leq i < k$, let $\pi'[x_1, x_0]_{i}$ denote the unique simple walk between $\pi(x_{i+1})$ and $\pi(x_i)$ in $t$.

The backbone of $\rho$ is the walk

$$\pi'[x_1, x_2, \pi'[x_2, x_3, \pi'[x_3, \ldots, \pi'[x_{k-1}, x_k].$$

Note that in general the backbone of an accepting run is not a simple walk in $t$.

Roughly speaking, the backbone of an accepting run describes all the essential information of the run. Thus the nonemptiness of a local unary PA is reduced to the existence of an “accepting” backbone.

**Example 5.10** The backbone of an accepting run for the PA in Example 5.2 is the (simple) walk consisting of the downward path of $b$-nodes. Each $\pi'[x_1, x_2, \ldots, x_k]$ is a child-edge from one $b$-node to the next $b$-node.

We define a unary PA $A' = (Q', I', F', \gamma', \Delta')$ from $A$ as follows.

- $Q' = \{(y, q, x) \mid x, y \in \text{Free}(\gamma(q))\}$,
- $I' = \{(y, q, x) \mid q \in I, F' = \{(y, q, x) \mid q \in F\}$,
- For every state $(y, q, x) \in Q'$, the invariant $\gamma'(y, q, x)$ is defined as follows. Let first $P'_0$ be the data pattern obtained from $\gamma(q)$ by applying successively the following operation until no more new constraints are obtained:
  - if $x \sim y$ and $y \sim z$ are atomic formulas, then $x \sim z$ is added,
  - if $x \sim y$ and $y \sim z$ are atomic formulas, then $x \sim z$ is added.

The invariant $\gamma'(y, q, x)$ is now defined as $P'_0$ restricted to the set of nodes consisting of all ancestors of $x, y$ in $\gamma(q)$, and of all nodes on the unique simple walk between $x$ and $y$ in $\gamma(q)$.

- $((y_1, q, x_1), (y_2, q', x_2)) \in \Delta'$ if $(q, q', x_1, y_2) \in \Delta$ and $y = y_2$.

**Remark 5.11.** Note that each state invariant $\gamma'(y, q, x)$ in $A'$ is either a tree consisting of exactly a path (if $x$ is the ancestor of $y$ or vice versa), or a tree consisting of three
paths sharing a common endpoint (if neither \(x\) is the ancestor of \(y\) nor \(y\) is the ancestor of \(x\)). In the PA of Example 5.2 all invariants \(\gamma'((y,q,x))\) are a single child-edge between two \(b\)-nodes.

The following lemma is shown by completing every accepting run of \(A'\) to an accepting run of \(A\). This is possible, since every matching of state invariants in \(A'\) can be lifted to a matching of state invariants in \(A\) by adding subtrees.

**Lemma 5.12.** \(A\) is nonempty iff \(A'\) is nonempty.

In the following, we let \(r\) denote the maximum size of the state invariants of \(A'\).

**Bounding the number of branches**

We introduce some additional notations. Let

\[
\rho = (x_0,1,0,0,2,0) \mid (x_0,1,1,0,1,2) \mid (x_1,1,1,1,2,2)
\]

be an accepting run of \(A'\) over a data tree \(t\). As in the definition of backbones of the accepting runs of \(A\), we can define the simple walks \(\pi[i,1,1,2]\) for every \(i : 0 \leq i \leq k\). Suppose that \(n\) is a node in \(t\) and \(i : 1 \leq i \leq k\). The simple walk \(\pi[i,1,1,2]\) is said to be around \(n\) if \(n\) appears in \(\pi[i,1,1,2]\).

**Lemma 5.13.** If \(A'\) is nonempty, then it accepts a data tree where each node has at most exponentially many children.

**Proof.** (sketch) Suppose \(A'\) is nonempty and

\[
\rho = (x_0,1,0,0,2,0) \mid (x_0,1,1,0,1,2) \mid (x_1,1,1,1,2,2)
\]

is an accepting run of \(A'\) over a data tree \(t\).

By a technical argument, we can show that there exists another tree \(t'\) and an accepting run \(\rho'\) over \(t'\) such that any node \(n\) in \(t'\) has exponentially many children. The idea is to consider all simple walks \(\pi[i,1,1,2]\) around \(n\) in \(\rho\), say \(\pi[i_1,1,1,2], \ldots, \pi[i_k,1,1,2]\) (where \(0 \leq i_1 < \cdots < i_k \leq k\)), and modify \(t\) and \(\rho\) such that \(\rho'\) has only exponentially many walks around \(n\). For this we consider the set \(D_c(n)\) of data values occurring in ancestors of \(n\) at distance at most \(r\), together with data values in \(\pi[i,1,1,2]\) and \(\pi[i,1,1,2]\) around \(n\) have the same (data-free) structure, contain the same values from \(D_c(n)\), and have the same data equality relation among the nodes, then we can use them, renaming some data values, in order to get a run \(\rho'\) with less simple walks around \(n\). The argument is iterated until there are only exponentially many simple walks around \(n\). Details can be found in the extended version.

Because only the simple walks around \(n\) in \(\rho'\) may visit the children of \(t'\), it follows that the arity of \(n\) in \(t'\) can be assumed to be at most exponential. This argument is applied bottom-up for every node \(n\).

From Lemma 5.13, it follows that \(A'\) can be seen as a unary PA over ranked data trees of exponential arity.

In the following, we restrict our discussion to ranked data trees of exponential arity.

**Bounding the number of data values**

**Lemma 5.14.** If \(A'\) is nonempty (over ranked data trees of exponential arity), then it accepts a data tree containing only exponentially many data values.

**Proof.** Recall that \(r\) is the maximum size of the state invariants of \(A'\). For a ranked data tree \(t\) and a node \(n\) in \(t\), let \(t'_n\) denote the subtree of \(t\) rooted at \(n\), with all the descendents of \(n\) at distance (strictly) greater than \(r\) from \(n\) removed. The profile of \(t'_n\) is the tree \(t'_n\), with the data values removed and replaced by the data value (in)equality relations \(\sim\), \(\approx\) (restricted to \(t'_n\)).

Because all the state invariants in \(A'\) are local patterns, it is not difficult to observe that

- If two data trees \(t, t'\) of the same tree structure satisfy that for every node \(n\), \(t'_n\) and \(t'_n\) have the same profile, then \(t\) is accepted by \(A'\) iff \(t'\) is accepted by \(A'\).

From the fact that there are at most exponentially many data values in each of those \(t'_n\) (this is due to the fact that a \(2^r\)-ary ranked tree of depth at most \(r\) has \(2^{O(r^2)}\) nodes), it follows that if \(t\) is a data tree accepted by \(A'\), then some data renaming can be applied bottom-up to \(t\), and a data tree \(t'\) can be obtained so that for every node \(n\), \(t'_n\) and \(t'_n\) have the same profile, moreover, \(t'\) contains only exponentially many data values. According to the above observation, it follows that \(t'\) is accepted by \(A'\).

**TWA over ranked trees**

From Lemma 5.14, it follows that for the nonemptiness problem we can restrict our attention to data trees containing exponentially many data values. By viewing the exponentially many data values as additional labelings, \(A'\) can be seen as a TWA \(B\) over ranked (non-data) trees. Because \(B\) needs to remember only polynomially many such labelings (data values), in order to check the data constraints in state invariants of \(A'\), it follows that \(B\) has exponentially many states.

On the other hand, it is known that the nonemptiness of TWA over ranked trees can be solved in exponential time [6]. Thus it follows that the nonemptiness of \(A'\), thus the nonemptiness of \(A\), can be solved in doubly exponential time.

**Local runs for unary PA where state invariants are not local patterns**

Finally to decide the local accepting runs of a unary PA \(A\), we consider the slightly more general case that the state invariants of \(A\) are not necessarily local patterns. In this situation, for each state \(q\) with the state invariant \(P\) in \(A\), there is a variable \(x\) in \(P\) such that for every transition into or out of \(q\), if a variable \(y\) in \(P\) is used in the transition invariant of the transition, then \(y\) belongs to the same component of \(P\) as \(x\). Therefore, for each state \(q\) with the state invariant \(P\), we are able to first ignore the variables of \(P\) not in the same component as \(x\) and get a unary PA \(A_1\) whose state invariants are all local patterns. Moreover, it is easy to see that \(A\) is nonempty iff \(A_1\) is nonempty, since from any data tree \(t\) accepted by \(A_1\), we can append some additional subtrees.
to \( t \) and get a tree \( t' \) accepted by \( A \). Because all the state invariants of \( A_1 \) are local patterns, it follows that the algorithm stated above can be used to solve the nonemptiness of \( A_1 \), thus also the nonemptiness of \( A \).

**Nonemptiness for arbitrary unary PAs**

Let \( A = (Q, I, F, \gamma, \Delta) \) be a unary PA. By the arguments for the local accepting runs of unary PAs, we can first compute all tuples \( (q, q') \) with \( q, q' \in Q \) such that there is a local run of \( A \) from \( q \) to \( q' \) over some data tree \( t \). Actually we need a more precise computation. We say that a run \( \rho \) starts in state \( q \) at the \( i \)-th component if \( \rho = (q, \pi)(q, \psi, q_1) \cdots \) and the variable \( x \in \text{Free}(\gamma(q)) \) of \( \psi \) is mapped by \( \pi \) to the \( i \)-th component of \( \gamma(q) \). Similarly, we say that a run ends in state \( q' \) at the \( j \)-th component. It is clear from the arguments for local accepting runs of unary PAs that all tuples \( (q, q', i, j) \) such that there is a data tree with a local run starting at \( q \) at the i-th component and ending at \( q' \) at the j-th component, can be computed in \( 2 \text{EXPTIME} \). Let \( X \) be the set of all such tuples.

Similarly to the proof of Proposition 5.8, we claim that \( A \) is nonempty iff there exist tuples

\[
(q_1, q'_1, i_1, j_1), \ldots, (q_k, q'_k, i_k, j_k)
\]

in \( X \) satisfying the following:

- there exist global runs from some \( q_0 \in I \) to \( q_1 \), and from \( q_k \) to some \( q_f \in F \),
- for every \( 1 \leq l < k \), either there is a global run from \( q'_l \) to \( q_{l+1} \), or otherwise \( q'_l = q_{l+1} \) and \( j_l \neq i_{l+1} \).

The “only if” direction is clear, since every run can be decomposed as above. For the “if” direction we show as in Proposition 5.8 how to build a data tree together with an accepting run of \( A \). Suppose that a data tree \( t' \) together with a run on \( t' \) from \( q_0 \) to \( q'_1 \) have been defined. If there exists a global run from \( q'_l \) to \( q_{l+1} \), then we proceed as before; if \( q'_l \neq q_{l+1} \), then we attach a data tree \( t_0 \) witnessing the local run for \( (q_{l+1}, q'_l, i_{l+1}, j_{l+1}) \) at a proper place of \( t' \) according to the relative position of the \( j_l \)-th component and \( i_{l+1} \)-th component in \( \gamma(q'_l) \), by renaming some data values in \( t_0 \) according to \( \gamma(q'_l) \).

**Lower bound**

For the lower bound, we show a polynomial-time reduction from \( \text{PSPACE} \) alternating Turing machines to the nonemptiness of unary PA over data trees. Then the lower bound follows from the equivalence of \( \text{EXPTIME} \) and \( \text{APSPACE} \).

Let \( M \) be a polynomial-space alternating Turing machine and \( w \) be an input of length \( m \). We can construct a unary PA \( A \) to simulate the computation of \( M \) on \( w \) as follows. Each configuration in the computation of \( M \) over \( w \) is encoded by a path of polynomial length \( m' = \rho(m) \) and the computation of \( M \) over \( w \) is encoded into a tree, by representing configurations vertically.

The automaton \( A \) needs to traverse a (sub)tree representing an accepting computation tree of \( M \). If the current configuration \( C \) is existential then \( A \) guesses a branch to go down. If \( C \) is universal, then a first guess downwards is done, and when returning to \( C \) from below, the next guess downwards needs to be done (we assume that every configuration has 0 or 2 successors, and that we see at the beginning of the encoding if it is the first or the second successor).

Suppose that from \( C \) the automaton went down into a branch corresponding to \( C' \). The automaton checks that \( C \vdash C' \) as follows: going down on \( C' \) it checks the \( i \)-th position of \( C' \) w.r.t. the \( i \)-th position in \( C \). Notice that the latter is the (unique) position located at distance \( m' + 1 \) upwards, so this can be expressed by the state invariants and the transition constraints (formally we have a disjunction on polynomially many cases accounting for the symbols that should occur at some in \( C' \) and \( C \), respectively; this can be handled by nondeterministic transitions of \( A \)).

**Remark 5.15.** Over data words, we can show that the upper and lower bound match, and the nonemptiness problem of unary PA is \( \text{PSPACE} \)-complete.

Moreover, if we assume that all the data tree patterns in a unary PA \( A \) are over the schema \( S^+ \), instead of \( S^* \), then we basically have no data values, since all values can be made equal. So we have to consider TWA over unranked trees and the complexity can be shown to be \( \text{EXPTIME} \)-complete.

**6. RELATED WORK**

The datalog language has been studied in depth for twenty five years. The emptiness problem of datalog is decidable [1]. On the other hand, the containment problem is in general undecidable [16]. As already stated, decidability is obtained in two restricted settings: (i) the problem of containment of datalog programs in non-recursive datalog programs [9] and (ii) the containment problem for monadic datalog programs [10]. The first problem is \( 3 \text{EXPTIME} \)-complete and the second is \( 2 \text{EXPTIME} \)-complete. The lower bound of the second problem is proved in [3]. It is shown in [8, 3] that the containment for monadic datalog programs becomes more tractable under some restrictions.

Query languages over data trees have been studied for less than a decade. Some restrictions of the language XPath over unranked ordered data trees have been notably studied. In general, XPath does not allow recursive operators other than the descendant relation and restricts the use of the node equality tests. In general, the problem of the satisfiability of XPath queries is undecidable over unranked ordered data trees [4]. Decidability for some restricted fragments of the language is shown in [13]. Other decidable fragments of logic over data trees with restricted data value comparisons have been studied. In particular, the satisfiability of the fragment of first-order logic over data trees using only two variables is decidable [7].

The language XPath can be generalized to tree pattern queries or conjunctive queries over unordered data trees that allow data equality test. [12] studies some kind of tree patterns that have to be mapped to the tree by an injective homomorphism. The problem of the containment of these tree patterns with equality and inequality tests is undecidable. However it becomes decidable over bounded depth trees. The containment of conjunctive queries over unordered data trees under regular language constraints is considered in [5]. They show the decidability of the fragment where only data value equalities are allowed and the undecidability when both data value equalities and inequalities are allowed.

Datalog over trees have been originally considered in [14]. They show that monadic datalog over unranked ordered trees with the signature first-child, next-sibling, last-child is equivalent to monadic second order logic on the same signature. A query language based on recursive patterns is
introduced in [2]. They show that for some restricted form of data value equalities, the problem of containment of two recursive patterns is decidable.

7. CONCLUSION

We have presented an in-depth study of the analysis of datalog programs over data trees. We have shown that the structure of data trees greatly complicates the problem. We have demonstrated that on the other hand, bounding the depth of the trees simplifies the problem. We have also introduced the new notion of pattern automata that roughly correspond to linear datalog programs and used them to obtain a novel decidability result in presence of data inequalities.

Some important questions remain open.

• One can consider restrictions on the use of navigational operators:
  - It is proved in [14] that monadic datalog over trees without data and MSO over trees are equivalent in absence of the relation desc. This result leads to the decidability of the containment problem for monadic datalog programs over trees using only the relation child (and not desc). The problem is still open when desc is considered.
  - The problem of containment of monadic datalog programs in unions of conjunctive queries over data trees is undecidable. This is for data trees with the operator desc. The question remains for programs without desc.

• One can consider restrictions on the use of data equalities. Indeed [2] shows that the containment problem, for a regular language of patterns with data equalities, is decidable when the use of data equalities is conained in some manner. It would be interesting to consider analogous restrictions in the context of the containment of monadic datalog programs in unions of conjunctive queries.

• By Corollary 5.6, the emptiness problem for linear monadic datalog$\sim$programs is decidable. On the other hand, by Proposition 3.1, the emptiness problem for linear binary datalog$\sim$programs is undecidable over data trees. The decidability of the emptiness problem for non-linear and monadic datalog$\sim$programs over data trees remains open.

8. REFERENCES