Logical Methods in Computer Science
Vol. 5 (1:4) 2009, pp. 1–21
www.lmcs-online.org

Submitted Dec. 20, 2006
Published Feb. 27, 2009

THE COMPLEXITY OF DATALOG ON LINEAR ORDERS

MARTIN GROHE [a] AND GOETZ SCHWANDTNER [b]

[a] Institut für Informatik, Humboldt-Universität zu Berlin, Unter den Linden 6, 10099 Berlin, Germany
e-mail address: grohe@informatik.hu-berlin.de

[b] Institut für Informatik, Johannes Gutenberg-Universität, 55099 Mainz, Germany
e-mail address: schwandtner@uni-mainz.de

Abstract. We study the program complexity of datalog on both finite and infinite linear orders. Our main result states that on all linear orders with at least two elements, the nonemptiness problem for datalog is EXPTIME-complete. While containment of the nonemptiness problem in EXPTIME is known for finite linear orders and actually for arbitrary finite structures, it is not obvious for infinite linear orders. It sharply contrasts the situation on other infinite structures; for example, the datalog nonemptiness problem on an infinite successor structure is undecidable. We extend our upper bound results to infinite linear orders with constants.

As an application, we show that the datalog nonemptiness problem on Allen's interval algebra is EXPTIME-complete.

1. Introduction

Datalog is the language of logic programming without function symbols. Datalog has been extensively studied in database theory (see, e.g., [2, 6, 7, 11, 12, 17, 18, 13]). In particular, the complexity of evaluating datalog queries has been determined: The data complexity is PTIME-complete, and the program complexity (also known as expression complexity) and combined complexity are both EXPTIME-complete [8, 23].

While previous work on datalog was concerned with datalog over finite structures, in this paper we are mainly interested in infinite structures. Infinite structures occur naturally in spatial and temporal reasoning (and spatial and temporal databases). In temporal reasoning, time is usually modelled as an infinite linear order, sometimes discrete and sometimes dense. This motivates our study of datalog on infinite linear orders. Let us remark that our results also apply to an interval based temporal reasoning, carried out, for example in Allen's interval algebra [3] (see Sec. 3).

When studying the complexity of datalog on infinite structures, we consider the structure as fixed, that is, we are interested in program complexity. Note that the result of a datalog query on an infinite structure may be infinite, thus we cannot hope to compute the full query result in finite time. A reasonable version of the query evaluation problem

1998 ACM Subject Classification: F.4.1, D.3.2, H.2.3.
Key words and phrases: datalog, complexity, infinite structures, linear orders, boundedness.

DOI:10.2168/LMCS-5 (1:4) 2009

that avoids this problem is the *datalog tuple problem*, which asks whether a given tuple of elements is in the result. However, even for the tuple problem there is the technical issue of how to represent the elements of the tuple and how to represent the structure itself. The simpler *datalog nonemptiness problem* asks if the result of a query is empty. It is well known that on finite structures, the nonemptiness problem is in EXPTIME, and as long as the structures have two elements that can be distinguished by a datalog program, it is EXPTIME-complete (see Sec. 4). It is easy to see that there are infinite structures where the nonemptiness problem is undecidable. An example is the structure with one infinite successor relation (see Sec. 4).

Our first main result (Theorem 6.4) states that on all linear orders (finite or infinite), the datalog nonemptiness problem is decidable in EXPTIME; for all linear orders with at least two elements it is EXPTIME-complete.

A problem that has received considerable attention in the datalog literature is the boundedness problem (see, e.g., [1, 11, 15]). A datalog program Π is *uniformly bounded on a class C of structures* if there is a number $b = b(\Pi, C)$ such that for all structures $\mathcal{A}$ in C, the computation of Π on $\mathcal{A}$ reaches a fixed point in at most b steps. The boundedness problem asks if a given program is uniformly bounded on the class of all finite structures; it was shown to be undecidable in [11].

Our second main result (Theorem 7.2) states that every datalog program is uniformly bounded on the class of all linear orders. This also leads to the decidability of the datalog tuple problem on linear orders, provided the linear order satisfies certain effectivity conditions.

Technically, both results are based on an analysis of the *distance types* of tuples processed in the evaluation of a datalog program. Types are a tool from model theory; the type of a tuple of elements records "definable" information about this tuple. Our distance types record information about the relative order of and the pairwise distances between elements of a tuple. The crucial technical fact underlying the results is that the whole computation of a datalog program on a linear order can be described in terms of a finite number of distance types that is bounded in terms of the program (independently of the structure).

In the last section of this paper, we show how to incorporate constants into the distance type concept to transfer our results to datalog over linear orderings with a finite number of constants, which may occur in the datalog programs in question.

As related results, let us mention recent results on the complexity of constraint satisfaction problems on infinite structures [4, 5, 19]. With some handwaving, the datalog nonemptiness problem may be viewed as a "recursive version" of constraint satisfaction problems.[1]

2. Preliminaries

2.1. **Datalog.** An *atom* is an expression of the form $P(x_1, \dots, x_k)$, where P is k-ary relation symbol and $x_1, \dots, x_k$ are variables. We admit 0-ary relation symbols.[2] In the following,

[1] Our actual starting point was an attempt to understand the complexity of constraint logic programming, which combines logic programming and constraint satisfaction. It turned out, however, that this complexity is dominated by the complexity of the "logic programming" part, which then led to our interest in datalog.

[2] A 0-ary relation either is empty, or it consists of the empty tuple ().

we abbreviate tuples $(x_1, \ldots, x_k)$ by $\bar{x}$. A *datalog rule* is an expression ρ of the form

$$P\bar{x} \leftarrow Q_1\bar{y}_1, \ldots, Q_m\bar{y}_m,$$

where $P\bar{x}$, $Q_1\bar{y}_1, \ldots, Q_m\bar{y}_m$ are atoms. The tuples of variables $\bar{x}, \bar{y}_1, \ldots, \bar{y}_m$ need not be disjoint, and variables may be occur several times in each tuple. Furthermore, the variables in $\bar{x}$ are not required to be among those in $\bar{y}_1, \ldots, \bar{y}_m$. The atom $P\bar{x}$ is the *head* of the rule and $Q_1\bar{y}_1, \ldots, Q_m\bar{y}_m$ is the *body*. A *datalog program* is a set of datalog rules. Relation symbols occurring in the head of a rule of a datalog program Π are called *intensional relation symbols* or *IDBs*; all other relation symbols are called *extensional relation symbols* or *EDBs*.

Datalog programs are interpreted over relational structures. A *vocabulary* is a finite set τ of relation symbols, each with a fixed *arity*. A structure $\mathcal{A}$ of vocabulary τ consists of a (finite or infinite) set A and a k-ary relation $R^{\mathcal{A}}$ for every k-ary relation $R \in \tau$. We say that a datalog program Π is *over* a structure $\mathcal{A}$ if the vocabulary of $\mathcal{A}$ contains all EDBs of Π and none of the IDBs. Π is a datalog program *over* a class C of structures if Π is a program over all $\mathcal{A} \in C$.

Let Π be a Datalog program over a structure $\mathcal{A}$. The *computation of* Π *over* $\mathcal{A}$ is carried out in stages, in which the interpretation of the IDBs is computed; the interpretation of the EDBs is given by $\mathcal{A}$ and remains fixed. Initially, all IDBs are interpreted by the empty set. In each stage, a rule ρ of Π is applied, and some tuples of elements of A are added to the interpretation of the IDB occurring in in the head of rule ρ. Formally, for every k-ary IDB R we define a sequence $(R_i^{\Pi,\mathcal{A}})_{i\geq 0}$ of k-ary relations on the universe A of $\mathcal{A}$. We let $R_0^{\Pi,\mathcal{A}} = \emptyset$ for all IDBs R. Suppose now we have defined $R_{i-1}^{\Pi,\mathcal{A}}$ for all IDBs R. In stage i, we choose a rule ρ, say, $P\bar{x} \leftarrow Q_1\bar{y}_1, \ldots, Q_m\bar{y}_m$. An *instantiation* of ρ at stage i consists of tuples $\bar{a}, \bar{b}_1, \ldots, \bar{b}_m$ of elements of A matching the lengths of the variable tuples $\bar{x}, \bar{y}_1, \ldots, \bar{y}_m$, such that

- If two variables of the rule are equal, then the corresponding elements of the tuples are equal as well. For example, if $x_r = y_{st}$ then $a_r = b_{st}$.
- For $1 \leq r \leq m$: If Q_r is an EDB, then $\bar{b}_r \in Q_r^{\mathcal{A}}$. If Q_r is an IDB, then $\bar{b}_r \in Q_{r\,(i-1)}^{\Pi,\mathcal{A}}$.

We let

$$P_i^{\Pi,\mathcal{A}} = P_{i-1}^{\Pi,\mathcal{A}} \cup \{\bar{a} \mid \text{there exist tuples } \bar{b}_1, \ldots, \bar{b}_m \text{ such that } \bar{a}, \bar{b}_1, \ldots, \bar{b}_m \text{ is an instantiation of rule } \rho \text{ at stage } i\}.$$

For all IDBs $R \neq P$, we let $R_i^{\Pi,\mathcal{A}} = R_{i-1}^{\Pi,\mathcal{A}}$. To turn this into a well-defined deterministic process, we cycle through the rules ρ of Π in some fixed order. It can be shown that the result of the computation does not depend on this order. (It will be convenient later to apply only one rule at each stage, that is why we set up the computation this way.)

Note that for all IDBs R and for all $i \geq 0$ we have $R_i^{\Pi,\mathcal{A}} \subseteq R_{i+1}^{\Pi,\mathcal{A}}$. The process either reaches a fixed point after finitely many stages, that is, there is an i_0 such that $R_{i_0}^{\Pi,\mathcal{A}} = R_i^{\Pi,\mathcal{A}}$ for all $i \geq i_0$, or it continues forever (recall that $\mathcal{A}$ may be infinite). In both cases, we let $R_\infty^{\Pi,\mathcal{A}} = \bigcup_{i\geq 0} R_i^{\Pi,\mathcal{A}}$. Then the $R_\infty^{\Pi,\mathcal{A}}$ form a fixed point of the computation, that is, further applications of the rules do not increase the relations. This is obvious if a fixed-point is reached in finitely many stages, but also easy to see if not. The result of the computation is the interpretation of the IDBs in this fixed point.

We usually write R_i^{Π} and R_∞^{Π} instead of $R_i^{\Pi,\mathcal{A}}$ and $R_\infty^{\Pi,\mathcal{A}}$ if $\mathcal{A}$ is clear from the context. For an easier reference, we define the following set of parameters for a datalog program Π:

By m_L we denote the maximal IDB arity (i.e. variables on the left hand side, head part of program rules), by m_R the maximal number of different variables occurring in a rule. By n_R we denote the number of rules of Π and by n_I the number of IDB symbols, by m_I the maximal number of IDB occurrences in a rule body. All these parameters are bounded from above by the length $n := |\Pi|$ of Π in some standard encoding.

For a more detailed introduction to datalog, we refer the reader to [2].

2.2. **Linear orders.** A *linearly ordered set* is a structure $\mathcal{A} = (A, <^{\mathcal{A}})$ of vocabulary $\{<\}$, where the binary relation $<^{\mathcal{A}}$ is a linear order of the universe A. For brevity, we refer to linearly ordered sets just as *linear orders*. Moreover, we usually omit the superscript in $<^{\mathcal{A}}$ and use the symbol $<$ to denote both the relation $<^{\mathcal{A}}$ and the relation symbol $<$. We write $a \leq b$ instead of ($a < b$ or $a = b$). The *distance* $d(a, b)$ between two elements $a < b \in A$ is the maximum $d \geq 0$ such that there are elements $c_0, \dots, c_d \in A$ with $a = c_0 < c_1 < \cdots < c_d = b$ if this maximum exists, and ∞ otherwise. The linear order $(A, <)$ is *dense without endpoints*, if for all $a \in A$ there are $b, c \in A$ such that $b < a < c$, and for all $a, b \in A$ with $a < b$ there is a $c \in A$ such that $a < c < b$.

We consider linear orders in the strict sense, that is, a linear order is always antireflexive. For orders in the sense of "less-than-or-equal-to", the datalog nonemptiness problem is trivial,[3] because we can always satisfy all atoms by interpreting all variables by the same element of the universe.

2.3. **Algorithmic problems.** We shall study the complexity of the following two decision problems for fixed structures $\mathcal{A}$:

Datalog nonemptiness problem over $\mathcal{A}$

Instance: Datalog program Π over $\mathcal{A}$, IDB P of Π.
Question: Is $P_\infty^{\Pi,\mathcal{A}} \neq \emptyset$?

Datalog tuple problem over $\mathcal{A}$

Instance: Datalog program Π over $\mathcal{A}$, k-ary IDB P of Π, k-tuple $\bar{a}$ of elements of $\mathcal{A}$ (for some $k \geq 1$).
Question: Is $\bar{a} \in P_\infty^{\Pi,\mathcal{A}}$?

For an infinite structure (with finite vocabulary and finite EDB and IDB arities, but infinite universe), the tuple problem bears some difficulties with regards to the representation of the input tuple and the accessibility of the structure. To deal with the first difficulty, whenever we consider the tuple problem we assume that the universe of the structure $\mathcal{A}$ is a decidable set of strings over some finite alphabet. Furthermore, for linear orders $\mathcal{A} = (A, <^{\mathcal{A}})$ we assume that it is decidable whether for elements $a, b \in A$ and a nonnegative integer k there exist $a_1, \dots, a_k \in A$ with $a <^{\mathcal{A}} a_1 <^{\mathcal{A}} \cdots <^{\mathcal{A}} a_k <^{\mathcal{A}} b$. Note that if this is undecidable, then the datalog tuple problem over $\mathcal{A}$ is also undecidable. Thus our assumption is just a restriction to the interesting cases of the problem.

Let us emphasise that these effectivity assumptions on the representation are only required when we study the datalog tuple problem. For the nonemptiness problem, we do not need to make any assumptions on the representation or decidability of $\mathcal{A}$ whatsoever.

[3] Actually, the problem is still PTIME-complete; it is equivalent to the datalog nonemptiness problem over a structure with one element, which is equivalent to the satisfiability problem for propositional Horn clauses.

3. Datalog on Allen's interval algebra

Allen's interval algebra, introduced in [3], is an algebra of relations over open intervals on the real line. These interval relations are built as unions from the 13 basic relations describing the pairwise relative end points of two intervals (x^-, x^+) and (y^-, y^+) as Table 1 (taken from [19]). The algebra of these 2^{13} relations is equipped with the operations *converse* (denoted by $\cdot^{-1}$), *intersection* $\cap$ and *composition* $\circ$.

The complexity of constraint satisfaction problems over Allen's interval algebra and variants has been extensively studied (see, e.g., [19, 20, 21]). Constraint satisfaction problems may be viewed as datalog nonemptiness problems for programs with a single nonrecursive rule. Here, we are interested in the complexity of full datalog over the interval algebra.

Table 1: The 13 basic relations of Allen's interval algebra. The obvious inequalities $x^- < x^+$ and $y^- < y^+$ of each case have been omitted.

Basic relation		Converse relation		Example	Endpoints
x precedes y	$\mathbf{p}$	y preceded by x	$\mathbf{p}^{-1}$	*xxx* *yyy*	$x^+ < y^-$
x meets y	$\mathbf{m}$	y met by x	$\mathbf{m}^{-1}$	*xxxx* *yyyy*	$x^+ = y^-$
x overlaps y	$\mathbf{o}$	y overlapped by x	$\mathbf{o}^{-1}$	*xxxx* *yyyy*	$x^- < y^- < x^+ < y^+$
x during y	$\mathbf{d}$	y includes x	$\mathbf{d}^{-1}$	*xxx* *yyyyyyy*	$y^- < x^-$, $x^+ < y^+$
x starts y	$\mathbf{s}$	y started by x	$\mathbf{s}^{-1}$	*xxx* *yyyyyyyy*	$x^- = y^-$, $x^+ < y^+$
x finishes y	$\mathbf{f}$	y finished by x	$\mathbf{f}^{-1}$	*xxx* *yyyyyyyy*	$y^- < x^-$, $x^+ = y^+$
x equals y	$\equiv$			*xxxxx* *yyyyy*	$x^- = y^-$, $x^+ = y^+$

Let $\mathcal{I}$ denote the structure whose universe consists of all open intervals on the real line, and whose relations are the relations of the interval algebra. We observe that datalog programs over $\mathcal{I}$ can easily be translated into programs over the linear order $(\mathbb{R}, <)$ and vice versa:

Lemma 3.1. *The datalog nonemptiness problem over $\mathcal{I}$ is LOGSPACE-equivalent to the datalog nonemptiness problem over $(\mathbb{R}, <)$.*

Proof. The reduction from the nonemptiness problem over $\mathcal{I}$ to the one over $(\mathbb{R}, <)$ is straightforward by replacing the interval variables by endpoint variables. Since we do not allow any equality relation to be used, we simulate equality by identifying variables.

For the other direction, we transform the program Π over $(\mathbb{R}, <)$ to Π' by replacing all atoms $x < y$ by $\mathbf{p}(x, y)$. Then Π' is satisfiable if and only if Π is satisfiable: If $\mathbf{p}(x, y)$ holds, then $x^- < y^-$ is satisfied. If on the other hand $x < y$ holds, then there are elements x^+ and y^+ such that $\mathbf{p}((x, x^+), (y, y^+))$ is satisfied, because the order $(\mathbb{R}, <)$ is dense.

Both reductions can clearly be carried out in logarithmic space. □

4. Lower Bounds

The hardness results in this section are either from [8], or they can fairly easily be proved by the techniques used in [8]. It is easy to see that for every finite structure $\mathcal{A}$, the datalog nonemptiness problem over $\mathcal{A}$ is in EXPTIME. For every structure $\mathcal{A}$ whose

universe contains at most one element, the nonemptiness problem is in PTIME. Conversely, for every structure $\mathcal{A}$, the datalog nonemptiness problem over $\mathcal{A}$ is PTIME-hard, because the satisfiability problem for propositional Horn clauses is equivalent to the nonemptiness problem for datalog programs with only 0-ary relation symbols. As soon as a structure contains two distinguishable elements, the nonemptiness problem becomes EXPTIME-hard. This will be made precise in Lemma 4.1 below. For the reader's convenience, we sketch the proof. It requires some preparation.

A *successor structure* is a structure $\mathcal{B} = (B, S^{\mathcal{B}}, N^{\mathcal{B}})$, where B is either finite or countably infinite, and for some enumeration $b_0, b_1, \ldots$ of B, the binary relation $S^{\mathcal{B}}$ consists of all pairs (b_i, b_{i+1}), and the unary relation $N^{\mathcal{B}}$ only contains the element b_0.

Assume, that in some structure $\mathcal{A}$ with universe A, we can *define* a successor structure. This means that there exists a datalog program Π with an m-ary IDB U, a $2m$-ary IDB S, and an m-ary IDB N such that the structure $\mathcal{B} = (B, S^{\mathcal{B}}, N^{\mathcal{B}})$ with $B = U_\infty^{\Pi,\mathcal{A}}$, $S^{\mathcal{B}} = S_\infty^{\Pi,\mathcal{A}}$, and $N^{\mathcal{B}} = N_\infty^{\Pi,\mathcal{A}}$ is a successor structure. Then a given Turing machine transition function can be translated to a datalog program defining the following IDB relations:

symbol$_\sigma(\bar{x}, \bar{y})$**:** In step $\bar{x}$ of the computation the tape cell $\bar{y}$ contains the symbol σ.
cursor$(\bar{x}, \bar{y})$**:** At instant $\bar{x}$ the cursor points to cell $\bar{y}$.
state$_s(\bar{x})$**:** In step $\bar{x}$ the Turing machine is in state s.
accept: The computation has reached an accepting state.

Here $\bar{x}$ and $\bar{y}$ range over elements of the defined successor structure $\mathcal{B}$ and hence can be viewed as encoding natural numbers, which are used to address time steps and tape cells. We may define auxiliary IDB relations ensuring the consistency of the simulation and encoding the input on the tape of the machine. Then we have a program, computable in logarithmic space from the machine encoding, whose IDB *accept* is derivable (and hence nonempty) if and only if a machine run accepts in a number of steps bounded by the size of the successor structure.

Lemma 4.1. *Given a structure $\mathcal{A}$ such that two relations $U_0, U_1 \subset \mathcal{A}^k$, $k \in \mathbb{N}$, can be defined by a datalog program on $\mathcal{A}$, such that*

$$U_0 \cap U_1 = \emptyset,\ U_0 \neq \emptyset,\ U_1 \neq \emptyset.$$

Then the datalog nonemptiness problem over $\mathcal{A}$ is EXPTIME-hard.

Proof. Without loss of generality we may assume that $\mathcal{A}$ actually contains two k-ary relations U_0, U_1 which are nonempty and disjoint. Hence we can use these relations as EDB predicates in a datalog program. We prove that any deterministic Turing machine computation on input x, with $|x| = n$ and time bound $t(x) = 2^m$ ($m = m(n)$ being a function with variable n) can be simulated by a datalog program with IDBs having at most $2 \cdot k \cdot m$ free variables.

The elements in U_0 are used as 0 and the elements in U_1 as 1 to build a successor structure of binary vectors of arity m, leading to a successor substructure with values in $[0..2^m]$. The details can be found in [8] with slight modifications.

The maximal arity of any IDB relations involved is $2 \cdot k \cdot m$, defining the successor between two m-tuples of entries that have arity k.

By the construction of the Turing machine simulation, any machine computation running at most 2^m steps can be simulated using datalog programs with maximal arity $2 \cdot k \cdot m$, which concludes the proof. □

Corollary 4.2. *For every linear order $\mathcal{A} = (A, <)$ with at least two elements, the datalog nonemptiness problem over $\mathcal{A}$ is EXPTIME-hard.*

Proof. Let U_0 be the binary relation $x < y$ and U_1 the converse $x > y$. □

Note that over infinite structures, the datalog nonemptiness problem can easily become undecidable. One of the simplest examples of an infinite structure where this happens is an infinite successor structure:

Proposition 4.3. *Let $\mathcal{B}$ be an infinite successor structure. Then the datalog nonemptiness problem over $\mathcal{B}$ is undecidable.* □

The proof is another straightforward Turing machine simulation.

5. Distance types

Types are a model theoretic tool that we shall use for dealing with datalog programs on infinite orders. We define an appropriate notion of type and prove a lemma that links them with the evaluation of datalog programs.

Definition 5.1.

(1) A *distance atom* is an expression of the form $x \leq_d y$, $-\infty \leq_d x$, or $x \leq_d \infty$, where x, y are variables and d is a nonnegative integer. We may write $<_d$ instead of $\leq_d$ for $d > 0$. A *distance type* is a finite set of distance atoms.

We write $\delta(x_1, \ldots, x_k)$ to indicate that the variables of the distance type δ are among $x_1, \ldots, x_k$. The set of all distance types with variables among $x_1, \ldots, x_k$ is denoted by $\Delta(x_1, \ldots, x_k)$.

(2) Let $\mathcal{A} = (A, <)$ be a linear order, $\bar{a} = (a_1, \ldots, a_k) \in A^k$, and let $\delta = \delta(x_1, \ldots, x_k)$ be a distance type. Then $(\mathcal{A}, \bar{a})$ *satisfies* δ (we write: $\mathcal{A} \models \delta(\bar{a})$),[4] if
- for all atoms $x_i \leq_d x_j \in \delta$, there are $b_0, \ldots, b_d \in A$ such that $a_i \leq b_0 < b_1 < \ldots < b_d \leq a_j$ (that is, $x_i \leq_d x_j$ is interpreted as $x_i \leq x_j$ and $d(x_i, x_j) \geq d$);
- for all atoms $-\infty \leq_d x_j \in \delta$, there are $b_0, \ldots, b_d \in A$ such that $b_0 < b_1 < \ldots < b_d \leq a_j$;
- for all atoms $x_i \leq_d \infty \in \delta$, there are $b_0, \ldots, b_d \in A$ such that $a_i \leq b_0 < b_1 < \ldots < b_d$.

A distance type δ is *satisfiable* if there is a linear order $\mathcal{A}$ and a tuple $\bar{a}$ such that $(\mathcal{A}, \bar{a})$ satisfies δ.

(3) The *rank* of a distance atom $t \leq_d u$ is d, and the rank of a distance type δ is the maximum of the ranks of all atoms it contains. The set of all distance types in $\Delta(x_1, \ldots, x_k)$ of rank at most d is denoted by $\Delta_d(x_1, \ldots, x_k)$.

(4) Let $\mathcal{A} = (A, <)$ be a linear order, $\bar{a} = (a_1, \ldots, a_k) \in A^k$, and $d \geq 0$. The *distance-d type of $\bar{a}$ in $\mathcal{A}$*, denoted by $\mathrm{tp}_d(\mathcal{A}, \bar{a})$, is the distance type that contains:
- for $1 \leq i, j \leq k$ with $a_i \leq a_j$ the distance atom $x_i \leq_c x_j$, where $c = \min\{d, d(a_i, a_j)\}$;
- for $1 \leq j \leq k$ the distance atom $-\infty \leq_c x_j$, where $c \leq d$ is maximum such that there exists $b_0, \ldots, b_c \in A$ with $b_0 < \ldots < b_c \leq a_j$;
- for $1 \leq i \leq k$ the distance atom $x_i \leq_c \infty$, where $c \leq d$ is maximum such that there exists $b_0, \ldots, b_c \in A$ with $a_i \leq b_0 < \ldots < b_c$.

[4] Another common terminology is to say that a type is *"realised"* instead of "satisfied".

(5) A distance type δ is *complete* if there exists a linear order $\mathcal{A}$, a tuple $\bar{a}$ with $\mathcal{A} \vDash \delta(\bar{a})$, and $d \geq 0$ such that for each pair (a_i, a_j) of entries of $\bar{a}$ satisfying $a_i < a_j$ there is precisely one distance atom $a_i \leq_c a_j$ in δ with $0 < c \leq d$, and for each pair (a_i, a_j) with $a_i = a_j$ there are distance atoms $a_i \leq_0 a_j$ and $a_j \leq_0 a_i$ in δ.

The set of all complete distance types with variables among $x_1, \dots, x_k$ is denoted by $\Gamma(x_1, \dots, x_k)$, and the set of all types in $\Gamma(x_1, \dots, x_k)$ of rank at most d is denoted by $\Gamma_d(x_1, \dots, x_k)$.

Example 5.2. An example for a distance type from $\Delta(x, y, z)$ is:

$$\delta = x <_3 y,\ y <_2 z$$

This type δ is satisfied for some elements $a_1, a_2, a_3 \in A$, which we assign to the variables $x = a_1$, $y = a_2$ and $z = a_3$, if there exist $b_1, b_2, b_3 \in A$ with

$$\begin{aligned} a_1 < b_1 < b_2 < a_2 \qquad & \text{to satisfy } x <_3 y \\ a_2 < b_3 < a_3 \qquad & \text{to satisfy } y <_2 z\ . \end{aligned}$$

The occurring ranks of the atoms in delta show $\delta \in \Delta_3(x, y, z)$. δ is not complete, since there is no distance atom containing x and z and no distance atom containing $-\infty$ or ∞.

Let us point out some subtleties of these definitions that may be confusing. A distance type need not be satisfiable, but a complete distance type must be satisfiable.[5] Even though the "constants" $-\infty$ and ∞ appear in distance atoms, they are not part of the datalog language, and we do not require linear orders to have a minimum or maximum. The semantics of the atoms $-\infty \leq_d x$, or $x \leq_d \infty$ is well-defined in all linear orders.

Note that $x \leq_1 y$ is equivalent to $x < y$ and that $x \leq_0 y \wedge y \leq_0 x$ is equivalent to $x = y$. A distance type of rank 1 only contains information about the relative order of the variables and about equalities between the variables, and not about their distances. Hence we call distance types of rank 1 *order types*.

It it is easy to see that it can be decided in polynomial time in the number of variables whether a distance type is satisfiable and whether it is complete.

Definition 5.3. Let γ, δ be distance types. Then γ *implies* δ if for all distance atoms $x \leq_d x'$ in δ there is a distance atom $x \leq_{d'} x'$ with $d \leq d'$ in γ.

Lemma 5.4.

(1) *Let $\gamma(\bar{x}), \delta(\bar{y})$ be distance types such that γ implies δ. Then all variables in $\bar{y}$ also appear in $\bar{x}$, and for every linear order $\mathcal{A}$ and every tuple $\bar{a}$ such that $\mathcal{A} \models \gamma(\bar{a})$, for the projection $\bar{b}$ of $\bar{a}$ to the variables in $\bar{y}$ we have $\mathcal{A} \models \delta(\bar{b})$.*

(2) *Let $\delta \in \Delta_d(\bar{x})$. Then for all linear orders $\mathcal{A}$ and all tuples $\bar{a} \in A^k$,*

$$\mathcal{A} \models \delta(\bar{a}) \iff \text{there exists a } \gamma \in \Gamma_d(\bar{x}) \text{ such that } \gamma \text{ implies } \delta \text{ and } \mathcal{A} \models \gamma(\bar{a}). \tag{5.1}$$

We omit the straightforward proof. Note that statement (2) of the lemma implies that every type can be written as a disjunction of complete types.

Recall that for each IDB P of a datalog program Π over some fixed structure $\mathcal{A}$, by P_i^Π we denote the interpretation of P after the ith stage of the computation of Π. In the following lemma we will show how to describe the stages by finite sets of distance types, but first we will have a look at a simple example.

[5] In model theory, it is common to define types as being satisfiable sets of formulas.

Example 5.5. Let Π be the following two-rule program defining IDBs P and Q:

$$P(x,y) \leftarrow x < z_1,\ z_1 < z_2,\ z_2 < y.$$
$$Q(x,y,z) \leftarrow P(x,y),\ y < w,\ w < z.$$

Applying the first rule to the empty IDB relations at the beginning, the resulting relation P_1^Π contains all tuples satisfying the distance type $x <_3 y$, since there have to be three distance atoms satisfied between the elements assigned to x and y.

Applying the second rule to this stage 1, this distance type is copied to the type describing the tuples in Q_2^Π and on y and z the type $y <_2 z$ is imposed, leading to the following type describing Q_2^Π:

$$\delta = x <_3 y,\ y <_2 z$$

For programs using recursion and more rules leading to some form of disjunction, a single distance type is not enough to describe a relation, but sets of types are needed.

Lemma 5.6. *Let $\mathcal{A} = (A, <)$ be an infinite linear order and Π a datalog program over $\mathcal{A}$.*

Then for each k-ary IDB P of Π and each $i \geq 0$ there is a finite set $\Theta(P,i) \subseteq \Gamma(x_1,\dots,x_k)$ such that for all $\bar{a} \in A^k$ it holds that

$$\bar{a} \in P_i^\Pi \iff \text{there is a } \theta \in \Theta(P,i) \text{ such that } \mathcal{A} \models \theta(\bar{a}).$$

Furthermore, the rank of all types in $\Theta(P,i)$ is bounded by $(m_R)^i$, where m_R denotes the maximal number of variables in a rule of Π as usual.

Proof. For $i \geq 0$, let $d_i := (m_R)^i$.

We prove the claim by induction on i. The induction base for $i = 0$ is obvious: We let $\Theta(P,0) = \emptyset$ for all IDBs P.

For the induction step $(i \to i+1)$, we consider the application of a rule

$$\rho:\ P(\bar{x}) \leftarrow P^1(\bar{y}_1),\dots,P^\ell(\bar{y}_\ell),\epsilon(\bar{y}),$$

where $P^1,\dots,P^\ell$ are IDBs and $\epsilon(\bar{y})$ is a list of EDB atoms with variables in $\bar{y}$. We view ϵ as a distance type of rank 1; this will enable us to unify some of the arguments below. Let $Z = \{z_1,\dots,z_m\}$ be the set of all variables occurring in ρ, and let $\bar{z} = (z_1,\dots,z_m)$. Note that $m \leq m_R$ and hence $m \cdot d_i \leq d_{i+1}$.

For $1 \leq j \leq \ell$, let $\theta_j \in \Theta(P^j,i)$. Assume that there is a $\gamma \in \Gamma_{d_i}(z_1,\dots,z_m)$ which implies $\theta_1,\dots,\theta_\ell$ and ϵ. Suppose $w_1,\dots,w_m$ is an enumeration of Z in the order imposed by γ, and let $e_0,\dots,e_m \geq 0$ such that γ contains the distance atoms

$$-\infty \leq_{e_0} w_1, \quad w_p \leq_{e_p} w_{p+1} \text{ for } p = 1,\dots,m-1, \quad w_m \leq_{e_m} \infty.$$

We define a new type $\gamma|_{\bar{x}}$ in the variables $\bar{x}$ as follows:

- Let x be a variable in $\bar{x}$, say, $x = w_p$. Then $\gamma|_{\bar{x}}$ contains the distance atom $-\infty \leq_d x$ for $d = \sum_{q=0}^{i-1} e_q$.
- Let x, x' be variables in $\bar{x}$, say, $x = w_i$ and $x' = w_j$. Suppose that $i \leq j$. Then $\gamma|_{\bar{x}}$ contains the distance atom $x \leq_d x'$ for $d = \sum_{r=p}^{q-1} e_r$.
- Let x be a variable in $\bar{x}$, say, $x = w_p$. Then $\gamma|_{\bar{x}}$ contains the distance atom $x \leq_d \infty$ for $d = \sum_{q=p}^{m} e_q$.

It is easy to see that $\gamma|_{\bar{x}}$ is a complete distance type in the variables $\bar{x}$. The rank of $\gamma|_{\bar{x}}$ is at most m times the rank of γ and hence bounded by $m \cdot d_i \leq d_{i+1}$. Therefore, $\gamma|_{\bar{x}} \in \Gamma_{d_{i+1}}(\bar{x})$.

Furthermore, it is easy to verify that every tuple $\bar{a}$ that satisfies $\gamma|_{\bar{x}}$ has an extension to an m-tuple $\bar{c}$ that satisfies γ.

We let $\Theta(P, i+1)$ be the union of $\Theta(P, i)$ with all types $\gamma|_{\bar{x}}$, where $\gamma \in \Gamma_{d_i}(\bar{z})$ such that γ implies ϵ and there exist $\theta_j \in \Theta(P^j, i)$, for $1 \leq j \leq \ell$, implied by γ. We claim that for all tuples $\bar{a}$ it holds that $\bar{a} \in P^{\Pi}_{i+1}$ if and only if there is a $\theta \in \Theta(P, i+1)$ such that $\mathcal{A} \models \theta(\bar{a})$.

To prove the forward direction of this claim, let $\bar{a} \in P^{\Pi}_{i+1}$. If $\bar{a} \in P^{\Pi}_{i}$, then by the induction hypothesis there is a $\theta \in \Theta(P, i) \subseteq \Theta(P, i+1)$ such that $\mathcal{A} \models \theta(\bar{a})$. Suppose that $\bar{a} \in P^{\Pi}_{i+1} \setminus P^{\Pi}_{i}$. Then there is a tuple $\bar{c}$ interpreting the variables $\bar{z}$, with projections $\bar{a}$ on the coordinates of the variables in $\bar{x}$, $\bar{b}_j$ on the coordinates of the variables in $\bar{y}_j$, and $\bar{b}$ on the coordinates of the variables in $\bar{y}$, such that $\bar{b}_j \in (P^j)^{\Pi}_{i}$ for $1 \leq j \leq \ell$ and $\mathcal{A} \models \epsilon(\bar{b})$. By the induction hypothesis, for $1 \leq j \leq \ell$ there is a type $\theta_j \in \Theta(P_j, i)$ such that $\mathcal{A} \models \theta_j(\bar{b}_j)$. Let γ be the complete distance-d_i type of $\bar{c}$. Then $\gamma|_{\bar{x}} \in \Theta(P, i+1)$, and $\mathcal{A} \models \gamma|_{\bar{x}}(\bar{a})$.

For the backward direction, suppose that a tuple $\bar{a}$ satisfies a type $\gamma'(\bar{x}) \in \Theta(P, i+1)$. If $\gamma'(\bar{x}) \in \Theta(P, i)$, then $\bar{a} \in P^{\Pi}_{i} \subseteq P^{\Pi}_{i+1}$ by the induction hypothesis. Otherwise, there is a complete type $\gamma \in \Gamma_{d_i}(\bar{z})$ and types $\theta_j \in \Theta(P^j, i)$ for $1 \leq j \leq \ell$ such that $\gamma' = \gamma|_{\bar{x}}$ and γ implies $\theta_1, \ldots, \theta_\ell, \epsilon$. Let $\bar{c}$ be an m-tuple satisfying γ such that the projection of $\bar{c}$ on the coordinates of the variables in $\bar{x}$ is $\bar{a}$. For $1 \leq j \leq \ell$, let $\bar{b}_j$ be the projection of $\bar{c}$ on the the coordinates of the variables in $\bar{y}_j$. Then $\mathcal{A} \models \theta_j(\bar{b}_j)$. By the induction hypothesis, $\bar{b}_j \in (P^j)^{\Pi}_{i}$. Let $\bar{b}$ be the projection of $\bar{c}$ on the coordinates of the variables in $\bar{y}$. Then $\mathcal{A} \models \epsilon(\bar{b})$. Putting everything together, we obtain $\bar{a} \in P^{\Pi}_{i+1}$. □

Remark 5.7. Note that actually we have proved a slightly stronger bound on the ranks of the types in $\Theta(P, i)$: Letting d_i be the maximum rank of all types in $\Theta(P, i)$, we have

$$d_{i+1} \leq m_R \cdot d_i \tag{5.2}$$

for all $i \geq 0$. Furthermore, whereas the numbers d_i may depend on the order in which we apply the rules of the program, the bound (5.2) holds for all orders.

The following example shows that the ranks of the types can increase during a computation in a way that can get quite complicated:

Example 5.8. Consider the following program consisting of rules ρ_1, ρ_2 and ρ_3, with $\bar{x} = (x_1, \ldots, x_5)$. We use the abbreviation $x_i <_2 x_j$ for $x_i < y, y < x_j$ omitting some body variable y in the definition of Π, which does not appear elsewhere in the rules.

$$\begin{aligned}
\rho_1 : P\bar{x} \ &\leftarrow x_1 <_2 x_2,\, x_2 <_2 x_3,\, x_4 <_2 x_5.\\
\rho_2 : P\bar{x} \ &\leftarrow x_1 < x_2,\, x_4 < z_2,\, z_3 < y_4,\, y_5 < x_5,\\
&\qquad\quad P(x_2, x_3, z_1, z_2, z_3),\, P(y_1, y_2, y_3, y_4, y_5).\\
\rho_3 : P\bar{x} \ &\leftarrow P(x_1, x_2, x_3, z_1, z_2), P(y_1, x_4, x_5, y_2, y_3).
\end{aligned}$$

The rule ρ_1 is an initialization rule which initializes all given types to $<_2$.

The rule ρ_2 introduces $x_1 <_1 x_2$, reuses existing types by copying and sums up some existing atoms from possibly different existing types.

This rule uses two recursive occurrences of the IDB P, which in our description of the application of this rule by types leads to the use of two (possibly different) types from the type set describing earlier stages of P^{Π}_{∞}. We denote the ranks of the distance atoms from these two types occuring in our computation, using $\bar{a} = (a_1, \ldots, a_5)$ for tuples from such stages, by:

Ranks in distance types of the earlier stages of P_∞^Π		
first occurrence of P	$a_1 <_{c_1} a_2$	$a_4 <_{c_2} a_5$
second occurrence of P	$a_1 <_{c_1'} a_2$	$a_4 <_{c_2'} a_5$

The rule application of ρ_2 using these distance atoms will then impose the following type on the body variables, where the distance atoms are given in the order of the variable appearance in the rule, ommiting the atoms containing the variables z_1, y_1, y_2 and y_3, not part of the result:

$$x_1 <_1 x_2,\ x_4 < z_2,\ z_3 <_1 y_4,\ y_5 <_1 x_5,\ x_2 <_{c_1} x_3,\ z_2 <_{c_2} z_3,\ y_4 <_{c_2'} y_5$$

To combine these types by eliminating non-head variables, we rearrange these atoms:

$$x_1 <_1 x_2,\ x_2 <_{c_1} x_3,\ x_4 <_1 z_2,\ z_2 <_{c_2} z_3,\ z_3 <_1 y_4,\ y_4 <_{c_2'} y_5,\ y_5 <_1 x_5$$

After the elimination of non-head variables, the following type is added to the type set of P:

$$x_1 <_1 x_2,\ x_2 <_{c_1} x_3,\ x_4 <_{c_2+c_2'+3} x_5$$

Rule ρ_3 copies some distance atoms for x_1, x_2, x_3 and transfers some $x_2 <_c x_3$ to $x_4 <_c x_5$ in the result. We conclude the example with the shortest program run leading to a fixed point, described by the ranks of the types $x_1 <_{d_1} x_2$, $x_2 <_{d_2} x_3$ and $x_4 <_{d_3} x_5$. We assume, that always the smallest ranks are chosen. Longer runs could lead to even bigger intermediate results, but will have the same final result.

step	rule	d_1	d_2	d_3	remarks
1	ρ_1	2	2	2	
2	ρ_2	1	2	7	
3	ρ_2	1	1	12	using tuples in line 2 and 1
4	ρ_3	1	1	1	using tuple in line 3 twice

6. Upper Bounds for datalog programs on orders

Now that we have a formal description of the IDB relations in this case, we will use the concept of discrete order types to show an upper bound for the datalog nonemptiness problem. But before, we transform the program into some normal form which integrates the possible order types into the program by creating disjoint copies of each IDB, each having a different order type and hence leading to disjoint relations.

Definition 6.1. A datalog program over linear orders $\mathcal{A}$ is *type-disjoint*, if for every k-ary IDB P there is a complete order type $\gamma_P \in \Gamma_1(x_1, \ldots, x_k)$ such that for all linear orders $\mathcal{A} = (A, <)$ and all tuples $\bar{a} \in P_\infty^\Pi$ it holds that $\mathrm{tp}_1(\mathcal{A}, \bar{a}) = \gamma_P$.

The *order type* of an IDB P in a type-disjoint program Π is the order type γ_P.

Lemma 6.2. *For every datalog program Π over linear orders there is a type-disjoint datalog program Π' over linear orders with the following properties:*

(1) *For every IDB P of Π there are IDBs $P_1, \ldots, P_{n_P}$ of Π' of pairwise distinct order types, such that for every linear order $\mathcal{A}$,*

$$P_\infty^\Pi = \bigcup_{j=1}^{n_P} (P_j)_\infty^{\Pi'}.$$

(2) $n'_I \leq n_I \cdot 3^{m_L^2}$, $n'_R \leq 3^{m_L^2 \cdot (m'_I+1)} \cdot n_R$, $m'_R = m_R$ *and* $m'_L = m_L$, $m'_I = m_I$, *where* n'_I, m'_R, m'_L, m'_I *are the parameters of* Π'.

Furthermore, the program Π' *can be computed from* Π *in exponential time.*

Proof. From each IDB P of arity r, we create $n_P = 3^{r^2}$ distinct copies $P_0, \ldots, P_{n_P-1}$, each having a different order type. For $i \in \{0, \ldots, n_P - 1\}$, let $(i_0, \ldots, i_{r^2-1})$ be the ternary representation of the number i, $i = \sum_{j=0}^{r^2-1} i_j \cdot 3^j$ with $0 \leq i_j < 3$ for all $j = 1, \ldots, n_P - 1$. Then we link to each new IDB P_i a distance-1-type γ_{P_i}, which consists of the following distance atoms:

$$\begin{aligned} x_{j_1} = x_{j_2}, &\quad \text{if } i_{j_1+j_2 \cdot r} = 0 \\ x_{j_1} <_1 x_{j_2}, &\quad \text{if } i_{j_1+j_2 \cdot r} = 1 \\ x_{j_2} <_1 x_{j_1}, &\quad \text{if } i_{j_1+j_2 \cdot r} = 2 \end{aligned}$$

So each combination of distance atoms for all pairs of variables will be present in some γ_{P_i}. After computing these distance types, we transform the program in two stages. First, we change the head IDBs to the new IDB set consisting of the distinct copies created as above for each IDB of Π: Each rule ρ with head atom $P\bar{x}$, is replaced by copies $\rho'_0, \ldots, \rho'_{n_P-1}$ with head $P_j\bar{x}$ with $j \in \{0, \ldots, n_P - 1\}$, and the body copied from ρ and extended by EDBs $x_{j_1} < x_{j_2}$ for each $(x_{j_1} <_1 x_{j_2}) \in \gamma_{P_j}$. In case of $(x_{j_1} = x_{j_2}) \in \gamma_{P_j}$, we replace all occurrences of x_{j_2} by x_{j_1} afterwards. This simulates the equality relation, which is not available as EDB or IDB relation.

Each of the rules $\rho'_0, \ldots, \rho'_{n_P-1}$ is then itself replaced by copies which instead of the body IDBs from Π, use the IDBs of Π':

In each rule ρ_r, say, $P_l\bar{x} \leftarrow Q_1\bar{y}_1, \ldots, Q_m\bar{y}_m, \epsilon(\bar{x}, \bar{y}_1, \ldots, \bar{y}_m)$, with ϵ being a sequence of EDBs, each IDB Q_j of Π has been converted to a set of IDBs $\{Q_{j\ell}\}$. From these sets we generate all possible combinations $(Q_{1\ell_1}, \ldots, Q_{m\ell_m})$ and create from each combination a rule of Π':

$$P_l\bar{x} \leftarrow Q_{1\ell_1}\bar{y}_1, \ldots, Q_{m\ell_m}\bar{y}_m,\ \epsilon(\bar{x}, \bar{y}_1, \ldots, \bar{y}_m)\ ,$$

where the sequence E of EDB atoms is left untouched.

After that, we directly eliminate a rule with an inconsistent order type. This can be done by viewing the rule as graph with the variables being the nodes and the order atoms being directed edges. A check for a directed cycle, which can be carried out in time polynomial in the rule length, shows if the order type is inconsistent.

Each tuple added to a stage in the evaluation of Π introduced by some rule ρ of Π has a complete distance 1 type, so there will be one of the copies of ρ, which can be applied to add this tuple. Conversely, the newly created rules of Π' may only add tuples, for which a rule in Π exists adding this tuple.

Each IDB of arity r is converted to not more than 3^{r^2} copies and hence $n'_I \leq n_I \cdot 3^{m_L^2}$. For each rule, we need all combinations of copies of the newly created IDBs, adding up to at most $n'_R \leq \left(3^{m_L^2}\right)^{(m_I+1)} \cdot n_R = 3^{m_L^2 \cdot (m_I+1)} \cdot n_R$. □

For type-disjoint datalog programs, the nonemptiness problem can be solved in a simple fashion, essentially disregarding any recursion in rules. In the following lemma, we construct an execution sequence s that will suffice to decide the datalog nonemptiness problem.

Lemma 6.3. *Let* Π *be a type-disjoint datalog program over an infinite linear order* $\mathcal{A} = (A, <)$. *Then there exist an* $i_s \leq n_I$ *and a sequence* $s = (\rho_0, \rho_1, \ldots, \rho_{i_s-1})$ *of rules, such*

that after applying ρ_i to stage i for $i = 0, \dots, i_s - 1$, the emptiness is determined, i.e. for all IDBs P it holds that

$$P_{i_s}^{\Pi} = \emptyset \quad \Rightarrow \quad P_{\infty}^{\Pi} = \emptyset \ . \tag{6.1}$$

Such a sequence s can be computed in time $n_R \cdot n_I$.

Proof. We create the sequence s by cycling through the rules n_I times, adding those rules to s, which change an empty IDB to nonempty, formally: $s = (\rho_0, \rho_1, \dots, \rho_{i_s-1})$ such that there exist IDBs $P_1, \dots, P_{i_s}$ with $(P_i)_i^{\Pi} = \emptyset$, and after applying ρ_i to $(P_i)_i^{\Pi}$, $(P_i)_{i+1}^{\Pi} \neq \emptyset$ for $i = 0, \dots, i_s - 1$.

We continue this process until no more rules can be applied to make an empty IDB nonempty, but this can happen at most n_I times, immediately leading to the time bound for the computation. Note that nonempty IDBs are never modified by s.

The crucial observation is that, in a type-disjoint program, it only depends on the nonemptiness of the IDBs in the body if a rule adds new tuples to the head IDB, and not on the actual content of the body IDBs. This follows from the fact that at each stage the content of the IDBs is a union of disjoint complete types by Lemma 5.6, and that the distance types are monotone by Corollary 7.3. Thus in an infinite order, we can always add all tuples of sufficiently large finite distances.

We now show property (6.1) by contradiction:

Let $U = \left\{ R \;\middle|\; R_{i_s}^{\Pi} = \emptyset \ \wedge \ R_{\infty}^{\Pi} \neq \emptyset \right\}$ be the set of IDBs changing to nonempty after s and assume $U \neq \emptyset$. Then for each $R \in U$ there exist an $i_R \in \mathbb{N}$ and a rule ρ_R with:

$$R_{i_R}^{\Pi} = \emptyset, \text{ and applying } \rho_R \text{ to } R_{i_R}^{\Pi} : \ R_{i_R+1}^{\Pi} \neq \emptyset \ .$$

Let $P \in U$ be the IDB with $i_P = \min \{ i_R \mid R \in U \}$. By the definition of U and by the choice of i, all $Q \in U \setminus \{P\}$ have to satisfy $Q_{i_R}^{\Pi} = \emptyset$. Since a rule can be applied if and only if all body IDBs are nonempty, the rule ρ_R cannot depend on them and can be applied in stage i_s leading to a sequence of rule applications making more IDBs nonempty, a contradiction to the construction of s. □

Theorem 6.4. *The datalog nonemptiness problem over linear orders $\mathcal{A} = (A, <)$ is EXPTIME-complete*

Proof. The proof is a combination of several earlier results. A datalog program Π can by Lemma 6.2 be converted to a type-disjoint program Π'. For this kind of program Lemma 6.3 gives us a method to check which IDB relations of Π' will be empty after an evaluation of Π'. Since Π' is type-disjoint, each IDB relation of the original program Π will occur here as a collection of IDBs of Π', which can easily be determined. Thus, the question "$P_{\infty}^{\Pi} = \emptyset$?" can be answered by checking the type sets of all corresponding IDBs of Π'. Beside the time for this check and the time for the conversion of the programs, the time for determining the empty IDB relations of Π' is part of the running time. Using Lemma 6.2 and 6.3 the time of this step can be bounded from above by $O(n_R \cdot 9^{m_L^2 \cdot (m_I+1)})$, altogether clearly in EXPTIME and with the earlier shown EXPTIME-hardness the claim follows. □

7. Boundedness

A datalog program Π is *bounded on a structure $\mathcal{A}$* if there is computation of Π on $\mathcal{A}$, that reaches a fixed point after finitely many stages. Of course, this concept of boundedness is nontrivial only on infinite structures. The main result of this section is that datalog

programs are bounded on linear orders. Actually, we prove a stronger result giving a uniform bound on the number of evaluation steps that computable from the size of the program and does not depend on the structure. This stronger result is even meaningful for finite linear orders.

Definition 7.1. Let Π be a datalog program over a structure $\mathcal{A}$.

(1) A *computation sequence* for Π is a sequence s of rules of Π to compute all IDB relations, i.e. a sequence of rules satisfying the following conditions:
   - If s is finite, then after applying s, no further rule application adds a tuple to the IDB relations.
   - If s is infinite, then each rule of Π will occur infinitely often.

(2) The *closure ordinal* of Π on $\mathcal{A}$, denoted by $\mathrm{cl}(\Pi, \mathcal{A})$, is the length of the shortest computation sequence for Π on $\mathcal{A}$ ($\mathrm{cl}(\Pi, \mathcal{A}) = \infty$, if all computation sequences are of infinite length).

(3) Π is *bounded on* $\mathcal{A}$ if $\mathrm{cl}(\Pi, \mathcal{A}) < \infty$.

Now let C be a class of structures such that Π is a program over C.

(4) The *uniform closure ordinal* of Π on C, denoted by $\mathrm{ucl}(\Pi, C)$, is the maximum of the closure ordinals $\mathrm{cl}(\Pi, \mathcal{A})$ for $\mathcal{A} \in C$, if this maximum exists, and ∞ otherwise.

(5) Π is *uniformly bounded on* C if $\mathrm{ucl}(\Pi, C) < \infty$.

Note that if Π is uniformly bounded on C, then it is bounded on all $\mathcal{A} \in C$, but that the converse does not necessarily hold.

Theorem 7.2. *Let Π be a datalog program over the class LO of linear orders. Then Π is uniformly bounded on LO. More precisely, there is a computable function $b : \mathbb{N} \mapsto \mathbb{N}$ such that for $n = |\Pi|$ it holds that*

$$\mathrm{ucl}(\Pi, LO) \leq b(n).$$

Our proof of Theorem 7.2 is based on a simplification of the distance type concept which we will discuss before the presentation of the main proof. The proof presented here is an extension of the proof of Theorem 6.4, first transforming the program Π in question to a type-disjoint version Π' by Lemma 6.2 and then creating the initialization sequence s as in Lemma 6.3. After this process, we may eliminate all then empty IDB relations. Each remaining IDB P may only contain tuples of one complete order type ϑ_P.

Let $\gamma \in \Gamma(x_1, \ldots, x_n)$ be a complete distance type. Observe that γ is completely determined by its underlying order type and the distances d imposed by the distance atoms $-\infty \leq_d x$, $x \leq_d x'$, $x \leq_d \infty$. We can describe the distances by a tuple $\bar{d}^\gamma = (d_1^\gamma, \ldots, d_k^\gamma)$ of length $k = 2n + \binom{n}{2}$ with nonnegative integer entries. We call $\bar{d}^\gamma$ the *rank vector* of γ. We define a partial order $\preceq$ on the complete distance types in $\Gamma(x_1, \ldots, x_n)$ by letting $\gamma \preceq \gamma'$ if γ and γ' imply the same order type and $d_i^\gamma \leq d_i^{\gamma'}$ for $1 \leq i \leq k$. Observe that $\gamma \preceq \gamma'$ if and only if γ' implies γ. The following corollary is hence an immediate consequence of Lemma 5.4(1):

Corollary 7.3. *Let $\mathcal{A} = (A, <)$ be a linear order, $\bar{a} \in A^k$, and $\gamma, \gamma' \in \Gamma(x_1, \ldots, x_k)$ such that $\gamma \preceq \gamma'$. Then*

$$\mathcal{A} \models \gamma'(\bar{a}) \implies \mathcal{A} \models \gamma(\bar{a}).$$

The crucial observation that we will exploit in the following is that the computation of a type-disjoint datalog program can be described entirely in terms of sequences of rank

vectors for the IDBs. This follows from Lemma 5.6 stating that the computation can be described in terms of complete types and the observation that for type-disjoint programs it suffices to consider the rank vectors, because the order types of the IDBs are fixed.

After applying the sequence s and after eliminating empty IDBs, for each IDB P, the set $\Theta(P, i_s)$ is described by exactly one such vector, since $|\Theta(P, i_s)| = 1$ after the initialization sequence which adds at most one type to the type set of each IDB. By Corollary 7.3, all tuples realizing a type γ' with $\gamma \preceq \gamma'$, also realize the weaker type γ. Hence increasing the size of an IDB relation P by adding new tuples (which realize a newly added type γ') is only possible if in all present types $\gamma \in \Theta(P, i)$ some atom rank of γ is greater than the corresponding rank in γ', i.e. $\gamma \not\preceq \gamma'$.

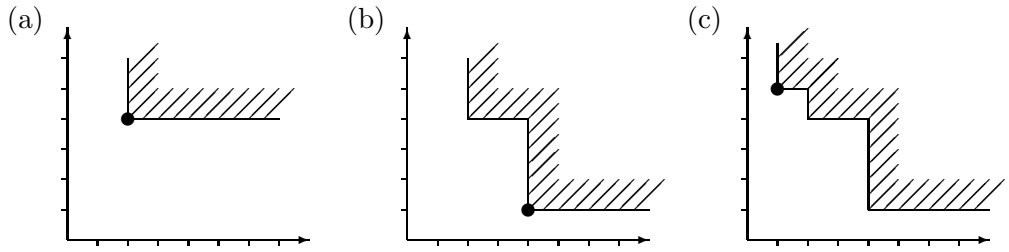
In terms of rank vectors, a type γ defines a set $\mathcal{H}_\gamma$ containing the vectors of all types γ' that are at least as restrictive as γ, i.e. $\gamma \preceq \gamma'$:

$$\mathcal{H}_\gamma = \left\{ (x_1, \ldots, x_{k_P}) \;\middle|\; x_\ell \geq d_\ell^\gamma \text{ for } \ell = 1, \ldots, k_P \right\}$$

Speaking of rank vectors, $\gamma \preceq \gamma'$ if the rank vector $\bar{d}^\gamma$ is *dominated* by the rank vector $\bar{d}^{\gamma'}$, i.e. for all $i = 1, \ldots, k_P$, $d_i^\gamma \leq d_i^{\gamma'}$. Then $\mathcal{H}_\gamma$ is the set of all types with a rank vector dominating the rank vector of γ.

Then creating a sequence of new types added to $\Theta(P, i)$ is equivalent to the search for a non-dominating sequence of rank vectors, where we call a (finite or infinite) sequence $x_1, x_2, \ldots$ *non-dominating* if for all i and j with $i < j$, x_j does not dominate x_i.

Figure 1 shows a graphical representation for $k_P = 2$. Figure 1 (c) shows a case where a new vector is added containing a coordinate greater than the maximum of all existing entries. But this growth can only occur in a limited manner, as we will show. Before, we introduce some notation.



Example of the description of an IDB relation with rank vectors of length 2 (x and y coordinate). Figure (a) shows a description with one rank vector, automatically including all types with rank vectors in the hatched area. Figure (b) shows the situation after a second rank vector was added, automatically including more types. In Figure (c), another vector is added.

Figure 1: Geometric Representation of a Type Set

Definition 7.4. Let $k \in \mathbb{N}$ and $\bar{x} = (x_1, \ldots, x_k) \in \mathbb{N}^k$. Then $||\bar{x}||_\infty := \max\{x_1, \ldots, x_k\}$. For $S \subset \mathbb{N}^k$, let $||S||_\infty := \max_{\bar{x} \in S} ||\bar{x}||_\infty$. Let $s_1, \ldots, s_\ell$ be finite sequences, each sequence consisting of tuples of some arity, and let $C = (s_1, \ldots, s_\ell)$ be a tuple of these sequences. Then $||C||_\infty := \max_{i=1}^{\ell} ||s_i||_\infty$, where the sequences are considered as sets.

To model the rank vectors occurring in the stages of the IDB relations, we introduce a corresponding sequence concept:

Definition 7.5 (**c-Bounded Run).** Let $t \in \mathbb{N}$, let $k_1, \dots, k_t \in \mathbb{N}$ and for $i = 1, \dots, t$ let $\bar{x}_i \in \mathbb{N}^{k_i}$. Let $c \in \mathbb{N}$. Then X is a *c-bounded run of* $(\bar{x}_1, \dots, \bar{x}_t)$, if

- $s_1^0, \dots, s_t^0$ are sequences of tuples, where for each i, s_i^0 consists of the tuple $\bar{x}_i$ only.
- The stage X_0 of X is the tuple $X_0 = (s_1^0, \dots, s_t^0)$.
- Inductively, the j-th stage $X_j = (s_1^j, \dots, s_t^j)$ of X is created from the stage X_{j-1} by choosing an $\ell \in \{1, \dots, t\}$, a $\mu_j \in \mathbb{N}$, and $\{\bar{x}_1, \dots, \bar{x}_{\mu_j}\} \subset \mathbb{N}^{k_\ell}$ such that
  - $\mu_j \leq (||X_0||_\infty \cdot c^{j-1})^{c^2}$
  - for $n \neq \ell$: $s_n^j = s_n^{j-1}$
  - $s_\ell^j = s_\ell^{j-1} \circ (\bar{x}_1, \dots, \bar{x}_{\mu_j})$ ($\circ$ meaning sequence concatenation)
  - s_ℓ^j is non-dominating
  - $||\bar{x}_i||_\infty \leq c \cdot ||X_{j-1}||_\infty$ for all $i = 1, \dots, \mu_j$

The condition on μ_j ensures that the sequence added in each stage is finite and bounded from above by some function of $||X_0||_\infty$, c and j, which will be needed for the computation of a uniform bound. The connection between the setting of datalog programs on orders and the c-bounded runs is given by the following lemma:

Lemma 7.6. *Let t be the number of nonempty IDB relations of the type-disjoint program Π' after the initialization sequence s of length i_s from Lemma 6.3. Then for each nonempty IDB relation P, the set $\Theta(P, i_s)$ contains exactly one rank vector. Let $\bar{d}_1, \dots, \bar{d}_t$ be these rank vectors. Let $m = \max\{m'_R, m'_I, m'_L\}$.*

(1) *For all $j = 1, \dots, t$: $||\bar{d}_i||_\infty \leq (m'_R)^{i_s} \leq (m'_R)^{n'_R n'_I}$.*
(2) *For each computation of Π' continuing the initialization sequence, the rank vectors added during this computation form an m-bounded run X of $(\bar{d}_1, \dots, \bar{d}_t)$.*

Proof. To prove (1), note that $||\bar{d}_i||_\infty \leq (m'_R)^{i_s}$ follows from Lemma 5.6 and $(m'_R)^{i_s} \leq (m'_R)^{n'_R n'_I}$ follows from Lemma 6.3.

(2) is proved by induction on the steps of the computation. Suppose at stage i of the computation of Π, a rule ρ with IDB P in its head is applied. Let $\gamma'_1, \dots, \gamma'_{\mu'}$ be the types in $\Theta(P, i+1) \setminus \Theta(P, i)$. It may be that some of the γ'_i dominate γ'_j for $j < i$ or $\gamma \in \Theta(P, i)$. We omit all these γ'_i and obtain a sequence $\gamma_1, \dots, \gamma_\mu$. Adding their rank vectors to the run obtained so far, we obtain a non-dominating sequence. The m-boundedness of the run follows from Remark 5.7. □

To show a computable uniform bound on c-bounded runs, we need two well known lemmas which we state here without proof:

Lemma 7.7 (**König's Tree Lemma (see, e.g.,** [16, 10])**).** *Let T be an* infinite *rooted directed tree with finite branching (i.e. each vertex has a finite number of children). Then T contains an infinite path starting at the root node.*

Lemma 7.8 (**Dickson's Lemma (see, e.g.,** [14, 9])**).** *All non-dominating sequences of tuples of natural numbers are finite.*

These finiteness (or infiniteness) properties allow us to compute a bound on the number of stages of c-bounded runs:

Lemma 7.9. *There is a nondecreasing computable function $f : \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ with the following property:*

For all $m \in \mathbb{N}$, $c \in \mathbb{N}$, $t \in \mathbb{N}$, $r \in \mathbb{N}$, $k_1, \dots, k_t \in \{1, \dots, r\}$ and $\bar{x}_i \in \mathbb{N}^{k_i}$ with $||\bar{x}_i||_\infty \leq m$, each c-bounded run of $(\bar{x}_1, \dots, \bar{x}_t)$ has at most $f(m, c, t, r)$ stages.

Proof. First, we have a look at an arbitrary choice of m, c, t, r, $k_1, \ldots, k_t$ and $\bar{x}_i$ (for $i = 1, \ldots, t$).

We create a labeled tree T containing all c-bounded runs of these values: The root node is labeled with X_0. Inductively, for each node labeled with a stage X_i, we create a child node for each stage X_{i+1} created from X_i and label it with the corresponding stage.

To create a stage X_{i+1} from X_i, we may choose each of the t sequences to extend it. Each sequence s^i_j has an arity k_j and by the last condition of a c-bounded run, the element added to this sequence may only have coordinates that are at most $c||X_i||_\infty$. Because of this and since the length of μ_{i+1} of the extension of the sequence in stage $i+1$ is bounded from above by $(||X_0||_\infty \cdot c^{j-1})^{c^2}$, there are only finitely many choices for a finite extension of a sequence and hence finitely many children for each node in this tree T (each for a different extension of some sequence).

A path in T (starting at the root node) corresponds to one c-bounded run. We now show that each path is finite: Assume, we have an infinite path p in T. This path p is labeled with the stages of a c-bounded run X. In each stage one of the sequences of X is extended by finitely many elements and since there are only t sequences there has to be one sequence that is extended in infinitely many stages. Each extension of this sequence is non-dominating, so we get an infinite non-dominating sequence. But by Dickson's Lemma each non-dominating sequence of tuples of natural numbers is finite, a contradiction. Hence all paths in T are finite.

Hence T has finite branching (only finitely many children to each node) and no infinite path. By König's Lemma T must be finite.

The height of T is the greatest number of stages that can occur in a c-bounded run of $\bar{x}_1, \ldots, \bar{x}_k$. Since T is finite, we can compute the whole tree and determine its height.

We discuss how to compute the value $f(m, c, t, r)$ for given values m, c, t and r:

For each fixed choice $k_1, \ldots, k_t$ of arities, the entries of all choices of the corresponding tuples $\bar{x}_1, \ldots, \bar{x}_t$ are bounded by m and thus for each tuple there are only finitely many choices. By computing the height of the tree (by creating the tree) to each choice of tuples one after the other and determining the maximum $h(k_1, \ldots, k_t)$, we have computed a bound on the number of stages for the c-bounded runs with sequence arities $k_1, \ldots, k_t$.

The parameter t determines the number of sequences in the runs considered and the parameter r limits the arities of these sequences. The maximum of the values $h(k_1, \ldots, k_t)$ over all possible r^t sequence arity tuples $(k_1, \ldots, k_t)$ is then the maximal number of stages in a c-bounded run with t sequences and sequence arities at most r and it can be computed by computing $h(k_1, \ldots, k_t)$ for all finitely many choices.

This maximum satisfies the properties of $f(m, c, t, r)$, and that f is nondecreasing is immediate: Increasing some parameter, all runs remain valid, but also longer runs may appear. □

This function will directly lead to the function b of Theorem 7.2:

Proof. (of Theorem 7.2) The program Π over a linear ordering $\mathcal{A} = (A, <)$ is first converted to an equivalent type-disjoint version Π' as in Lemma 6.2, which also gives the bounds $n'_I \leq n_I \cdot 3^{m_L^2}$, $n'_R \leq 3^{m_L^2 \cdot (m_I+1)} \cdot n_R$, $m'_R = m_R$ and $m'_L = m_L$ for the parameters of the new program Π'. Then the initialization sequence s as in Lemma 6.3 is determined, resulting in the first $i_s \leq n'_I \leq n_I \cdot 3^{m_L^2}$ stages.

While the empty relations of Π' can be neglected, each nonempty relation P of Π' has a type description $\Theta(P, i)$ with one rank vector each, and by Lemma 7.6 these rank

vectors satisfy the properties of an m_R-bounded run X. By Lemma 7.9, X has at most $f((m'_R)^{n'_R n'_I}, m'_R, n'_I, m'_L)$ stages.

We let $b(n) := f(n^{3^{n^4}}, n, n \cdot 3^{n^2}, n)$ and by the above bounds on the parameters and since each program parameter is bounded from above by the program length n, $f((m'_R)^{n'_R n'_I}, m'_R, n'_I, m'_L) \leq f(n^{3^{n^4}}, n, n \cdot 3^{n^2}, n) \leq b(n)$ and the claim follows.

Since $\mathcal{A}$ was chosen as arbitrary linear order, this bound also holds for $\mathrm{ucl}(\Pi, \mathrm{LO})$. □

It now follows easily that the datalog tuple problem is decidable on all linear orders, provided that the orders satisfy the effectivity conditions described in Section 2.3, which say that the elements of a structure are effectively represented, and that the distance-d type of a tuple can be computed.

Theorem 7.10. *The datalog tuple problem on linear orders is decidable.*

Proof. Using Lemma 5.6 and and Theorem 7.2, for each IDB we can compute the set of all complete types of tuples that are contained in an IDB-relation after the computation of the dalog program has been carried out. Then to decide whether a given tuple is contained in an IDB relation, we only have to check if it satisfies one of these types. □

The uniform closure ordinal of a datalog program can be also be used to decide the nonemptiness problem. By the EXPTIME-hardness of nonemptiness, it follows that the uniform closure ordinal of a datalog program over linear orders has to be at least singly exponential. We suspect that this lower bound is closer to the actual closure ordinal than our computable upper bound.

On dense linear orders without endpoints, we can match the singly exponential lower bound. Let *DLO* denote the class of dense linear orders without endpoints.

Theorem 7.11. *Let* Π *be a datalog program over the class of linear orders and* $n = |\Pi|$. *Then*

$$\mathrm{ucl}(\Pi, DLO) \leq 3^{n^2}.$$

Proof. Observe that on dense linear orders, distance types collapse to order types, because for all a, b and for all $d \geq 1$ we have $a < b \iff a \leq_d b$. So the only types to consider are distance-1-types (including equality atoms, when we consider complete distance-1-types) and the as type-disjoint version of a program as introduced in Lemma 6.2 contains all complete distance-1-types, it contains all types of interest for this case. After evaluating the initialization sequence as computed in Lemma 6.3, all complete distance-1-types describing the IDB relations are computed and hence no rule can be applied after that to add new types.

Since there at most 3^{n^2} different distance-1-types for a program of length n, the claim follows. □

8. Relational structures with constants

We may also consider datalog programs over linear orders $\mathcal{A} = (A, <, c_1, \ldots, c_r)$ with finitely many constants $c_1, \ldots, c_r$, each of them being interpreted as a fixed element of A, which may be used in the datalog programs in question.

To solve the datalog nonemptiness or tuple problem for such a program Π with constant symbols, we transform the program to a constant free version Π' by replacing each constant

c_i by a fresh variable and adding rule parts to transfer the values of all constants to all rules which are used during program execution. Using this technique, we show:

Theorem 8.1. *The datalog tuple problem on linear orders $\mathcal{A} = (A, <, c_1, \ldots, c_r)$ with finitely many constants (which may be used by the datalog programs) is decidable.*

Proof. We first transform the program Π over $\mathcal{A} = (A, <, c_1, \ldots, c_r)$ to a program Π' over the structure $\mathcal{A}' = (A, <)$ increasing the arity of each IDB symbol by r, such that for each IDB P of Π with arity s and its corresponding IDB P' of Π', and each $\bar{a} = (a_1, \ldots, a_s) \in A^s$ the following holds:

$$\bar{a} \in P_\infty^{\Pi,\mathcal{A}} \qquad \Longleftrightarrow \qquad \bar{c}\bar{a} = (c_1^{\mathcal{A}}, \ldots, c_r^{\mathcal{A}}, a_1, \ldots, a_s) \in (P')_\infty^{\Pi',\mathcal{A}'} \tag{8.1}$$

This is established by replacing all occurrences of the constant c_i by a fresh variable C_i, for $i = 1, \ldots, r$. To ensure, that all rule applications share the same values for the constants, we augment each IDB P of Π by the additional variables $\bar{C} = (C_1, \ldots, C_r)$ and replace all occurrences of $P(\bar{x})$ by $P'(\bar{C}, \bar{x})$ — in the rule bodies and the rule heads. This forces the values of the variables $C_1, \ldots, C_r$ to be identical in the body and head of each rule and hence in the whole pogram. For example, if $\phi(c_1, \ldots, c_r, x_1, \ldots, x_m, y_1, \ldots, y_n)$ is the formula appearing in the body of the rule

$$P(x_1, \ldots, x_m) \leftarrow \phi(c_1, \ldots, c_r, x_1, \ldots, x_m, y_1, \ldots, y_n).$$

we translate this rule to:

$$P'(C_1, \ldots, C_r, x_1, \ldots, x_m) \leftarrow \phi(C_1, \ldots, C_r, x_1, \ldots, x_m, y_1, \ldots, y_n).$$

Now the original program Π and the modified version Π' satisfy condition (8.1).

This transformation can be carried out in logarithmic space, since the number r of constants does only depend on the structure $\mathcal{A}$, not the input $(\Pi, P, \bar{a})$ of the tuple problem. The above construction is a logspace reduction from the tuple problem over linear orders with constants to the tuple problem over linear orders without constants, mapping the input $(\Pi, P, \bar{a})$ to the instance $(\Pi', P', \bar{c}\bar{a})$, increasing m_L and m_R by r and the total program size by a linear factor. For this constant free version of the tuple problem, Theorem 7.10 shows us how to solve it, calculating the type sets introduced in Lemma 5.6. □

We can also use these type sets computed in the above proof for solving the nonemptiness problem on $\mathcal{A} = (A, <, c_1, \ldots, c_r)$:

Corollary 8.2. *The datalog nonemptiness problem on linear orders $\mathcal{A} = (A, <, c_1, \ldots, c_r)$ with finitely many constants is decidable.*

Proof. On input (Π, P), we first calculate the type sets for the modified version Π', in which the constants have been replaced by the variables $C_1, \ldots, C_r$ as above. Then we instantiate the variables $C_1, \ldots, C_r$ by the constants of $\mathcal{A}$ in all types in $\Theta(P', \infty)$ for the IDB P' of Π' corresponding to IDB P of Π, each C_i by $c_i^{\mathcal{A}}$. Types which are then no longer satisfiable are deleted from the set. If and only if there are satisfiable types remaining, $P_\infty^{\Pi,\mathcal{A}}$ is nonempty. □

An intuitive argument, why the upper bound for the datalog nonemptiness problem on orders with constants is much higher than the complexity of the case without constants is, that as soon as constants are involved, the solution process has to consider distances between constants. A solution may for some two constants $c_i < c_j$ require a number of elements to be present in $\mathcal{A}$ between c_i and c_j, which is higher than the actual number of elements between c_i and c_j in $\mathcal{A}$. This case is only handled correctly by using distance types, order types alone do not suffice.

However, on dense linear orderings we may do better and match the EXPTIME lower bound:

Corollary 8.3. *The datalog nonemptiness problem on dense linear orders $\mathcal{A} = (A, <, c_1, \ldots, c_r)$ with finitely many constants can be decided in exponential time.*

Proof. This result is a straightforward combination of the preceeding proof and Theorem 7.11. □

Note that even though we use the decidability of the datalog tuple problem to prove the decidability of the datalog nonemptiness problem for linear orders with constants, we do not need to make any effectivity assumptions on the linear order $\mathcal{A}$ (cf. Sec. 2.3) here. The reason is that the constants are fixed in advance as part of the structure and thus all information about them can be hardwired into the algorithm.

9. Concluding remarks

We studied the complexity of datalog on linear orders. We precisely determined the complexity of the datalog nonemptiness problem: It is EXPTIME-complete on all linear orders with at least two elements. We also obtained a computable uniform upper bound on the closure ordinal of datalog programs on linear orders. Then best lower bound we know for the uniform closure ordinals is singly exponential.

The upper bound on the closure ordinals can be used to prove that the datalog tuple problem is decidable on computable orders leading to the same complexity bound for the nonemptiness and tuple problems on orders with constants. Based on these results, an implementation of the distance type concept for calculations seems feasible for applications, e.g. in temporal and spatial reasoning.

In his forthcoming PhD-thesis [22], the second author showed that most of the results obtained here can be extended to colored linear orders, that is, linear orders with additional unary predicates, and, at least partially, to colored trees, where trees are viewed as partial orders.

10. Acknowledgements

The authors thank Mark Weyer for proposing an elegant idea leading to the proof of Theorem 7.2.

References

[1] S. Abiteboul. Boundedness is undecidable for datalog programs with a single recursive rule. *Information Processing Letters*, 32:281–287, 1989.

[2] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[3] James F. Allen. Maintaining knowledge about temporal intervals. *Comm. ACM*, 26:832–843, 1982.

[4] M. Bodirsky and V. Dalmau. Datalog and constraint satisfaction with infinite templates. In B. Durand and W. Thomas, editors, *Proceedings of the 23rd Annual Symposium on Theoretical Aspects of Computer Science*, volume 3884 of *Lecture Notes in Computer Science*, pages 646–659. Springer-Verlag, 2006.

[5] M. Bodirsky and J. Nešetřil. Constraint satisfaction with countable homogeneous templates. *Journal of Logic and Computation*, 16:359–373, 2006.

[6] S. Chaudhuri and M.Y. Vardi. On the equivalence of recursive and nonrecursive datalog programs. In *Proceedings of the 11th ACM Symposium on Principles of Database Systems*, pages 55–66, 1992.

[7] S. S. Cosmadakis, H. Gaifman, P. C. Kanellakis, and M. Y. Vardi. Decidable optimization problems for database logic programs (preliminary report). In *Proceedings of the 20th ACM Symposium on Theory of Computing*, pages 477–490, 1988.

[8] Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and expressive power of logic programming. In *IEEE Conference on Computational Complexity*, pages 82–101, 1997.

[9] L.E. Dickson. Finiteness of the odd perfect and primitive abundant numbers with discitnct factors. *American Journal of Mathematics*, 3:413–422, 1913.

[10] R. Diestel. *Graphentheorie*. Springer, Berlin, second edition, 2000.

[11] H. Gaifman, H. Mairson, Y. Sagiv, and M.Y. Vardi. Undecidable optimization problems for database logic programs. *Journal of the ACM*, 40(3):683–713, 1993.

[12] G. Gottlob and Ch. Koch. Monadic datalog and the expressive power of web information extraction languages. *Journal of the ACM*, 51(1):74–113, 2004.

[13] G. Gottlob and C. H. Papadimitriou. On the complexity of single-rule datalog queries. *Inf. Comput.*, 183(1):104–122, 2003.

[14] W. Harwood, F. Moller, and A. Setzer. Weak bisimulation approximants. In *Proceedings of CSL'06*, LNCS, pages 365–379, Szeged, Hungary, 2006.

[15] Gerd G. Hillebrand, Paris C. Kanellakis, Harry G. Mairson, and Moshe Y. Vardi. Undecidable boundedness problems for datalog programs. *Journal of Logic Programming*, 25(2):163–190, 1995.

[16] Wilfrid Hodges. *A shorter model theory*. Cambridge University Press, New York, NY, USA, 1997.

[17] P.C. Kanellakis. Elements of relational database theory. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*. Elsevier Science Publishers B.v., 1990.

[18] Ph.G. Kolaitis and M.Y. Vardi. On the expressive power of datalog: tools and a case study. *Journal of Computer and System Sciences*, 51(1):110–134, 1995.

[19] Andrei Krokhin, Peter Jeavons, and Peter Jonsson. Constraint satisfaction problems on intervals and lengths. *Siam J. Discrete Math.*, 17(3):453–477, 2004.

[20] I. Meiri. Combining qualitative and quantitative constraints in temporal reasoning. *Artificial Intelligence*, 87:343–385, 1996.

[21] B. Nebel and Hans-Jürgen Bürckert. Reasoning about temporal relations: a maximal tractable subclass of allen's interval algebra. *Journal of the ACM*, 42, 1995.

[22] G. Schwandtner. *Datalog on Infinite Structures*. PhD thesis, Humboldt-Universität zu Berlin, 2008.

[23] M.Y. Vardi. The complexity of relational query languages. In *Proceedings of the 14th ACM Symposium on Theory of Computing*, pages 137–146, 1982.