ON PROBLEMS OF DATABASES OVER A FIXED INFINITE UNIVERSE

OLEG V. BELEGRADEK, ALEXEI P. STOLBOUSHKIN,
AND MICHAEL A. TAITSLIN

Abstract. In the relational model of databases a database state is thought of as a finite collection of relations between elements. For many applications it is convenient to pre-fix an infinite *domain* where the finite relations are going to be defined. Often, we also fix a set of domain functions and/or relations. These functions/relations are infinite by their nature. Some special problems arise if we use such an approach. In the paper we discuss some of the problems.

We show that there exists a recursive domain with decidable theory in which (1) there is no recursive syntax for finite queries, and in which (2) the state-safety problem is undecidable.

We provide very general conditions on the FO theory of an ordered domain that ensure collapse of order-generic extended FO queries to pure order queries over this domain: the *Pseudo-finite Homogeneity Property* and a stronger *Isolation Property*. We further distinguish one broad class of ordered domains satisfying the Isolation Property, the so-called *quasi-o-minimal* domains. This class includes all o-minimal domains, but also the ordered group of integer numbers and the ordered semigroup of natural numbers, and some other domains.

We generalize all the notions to the case of finitely representable database states — as opposed to finite states — and develop a general lifting technique that, essentially, allows us to extend any result of the kind we are interested in, from finite to finitely-representable states. We show, however, that these results cannot be transferred to arbitrary infinite states.

We prove that safe $Datalog^{\neg,<_z}$-programs do not have any effective syntax.

Contents

Date: May 30, 2003.

A part of this work had been done while Oleg Belegradek was visiting the Fields Institute for Research in Mathematical Sciences in Toronto (January–March, 1997).

This work of Alexei P. Stolboushkin has been partially supported by NSF Grant CCR 9403809.

A part of this research was carried out while M. A. Taitslin was visiting UCLA (partially supported by NSF Grant CCR 9403809), DIMACS and Princeton (partially supported by a grant from Princeton University). The work of the author was partially supported by the Russian Foundation of Basic Research (project code: 96-01-00086).

1. Introduction

1.1. **Infinite domains.** In the relational model of databases introduced by E.F. Codd [Cod70, Cod72] a database state is thought of as a finite collection of relations between elements. For example, the father – son relation can be represented in the form of one binary relation (or a two-column table). Names of the relations and their arities (numbers of argument places) are fixed and called a database scheme. Particular information stored in the relations of a given scheme is called a database state.

For instance, as we acquire more and more information about fathers and sons, the database states change, but the scheme (one binary relation) does not.

Database relations (tables) are always going to be finite.

Although relational databases were invented for finite collections of data, it is often convenient to assume that there is an infinite *domain* — for example, the integer or rational numbers or the strings — such that the data elements are chosen from this domain. Functions and relations defined over the entire domain, like $<$ and $+$, may also be used in querying, for example, if the language of first-order logic FO is used as the query language, its formulas may use database relations as well as the domain relations, while variables range over the entire domain.

These domain functions/relations are fixed (do not depend on a state, have the same meaning for any state) and are infinite by their nature. When we refer to a domain, we mean the domain together with the set of domain functions and relations that we consider.

1.2. **Finitely representable relations.** In the traditional relational database theory, the database relations are finite. The trouble with this is, answers yielded by relational queries may or may not be finite. This makes the traditional relational model not closed, in the sense that the output of queries is of a different nature than input. Kanellakis et al. [KKR90, KKR95] concentrated on the ordered domains of real, and rational numbers, and observed that, since the first order theories of these admit elimination of quantifiers, the answers to first order queries can be represented as quantifier-free first order formulas, and then, if we allow database relations to be arbitrary relations representable by quantifier-free first order formulas to begin with, the so modified relational model becomes closed in the above sense. Such relations are called *finitely representable* (for short, f.r.).

These finitely representable databases are a logical choice, because finitely representable relations appear as results of queries dealing with finite relations anyway, and it is also a natural choice in many applications, say, in geographical databases (cf. [KKR90, KKR95]).

1.3. **Safe queries.** In formulating queries to our database, we use a *query language.* The basic query language is the language of first-order logic (see [End72]). It uses domain functions/relations as well as the relations from the database scheme.

For example, consider the above database about fathers and sons. This database can be organized over the infinite domain of strings, and we throw in the equality $=$. Let F be the father – son relation. Then the formula $M(x)$:

$$\exists y, z(y \neq z \wedge F(x,y) \wedge F(x,z))$$

results in the unary relation (one-column table) that consists of those x's who have more than one son. While $G(x,z)$:

$$\exists y(F(x,y) \wedge F(y,z))$$

produces the table of "grandfather – son of his son".

Now we want the resulting relations (the answers to our queries) to be finite relations. The trouble with this is that often first-order formulas give infinite answers. Obviously, $\neg F(x,y)$ is such a formula. But worse than that, $M(x) \vee G(x,z)$ may give an infinite answer too, because $M(x)$ does not bound z at all.

The formulas (that *may* give infinite answers) are called *infinite,* or *unsafe,* as opposed to *finite,* or *safe*, formulas that always produce finite answers[1].

[1]Observe that the formula $M(x) \vee G(x,z)$ only gives an infinite answer if there is a man who parented two or more sons.

The situation was well understood in [Ull82] where J.D. Ullman raised the question of whether it is possible to tell safe formulas from unsafe. This has become known as the *safety problem.* This question was answered negatively by R.A. Di Paola [Di 69], M.Y. Vardi [Var81], and independently in [AGSS86]. The answer is negative for any infinite domain provided the database scheme contains at least one relation of arity > 1.

Although the formula that you use may be infinite, in a given state the answer may be finite. In this case, it would be desirable to get this finite answer. If the answer is infinite, it would be desirable to get this information, that the answer is infinite, from the database. Technically, the problem is, is it possible, for a *fixed* database state, to tell formulas with finite answers from those with infinite? This has become known as the *state-safety* problem [AGSS86]. By its very formulation this problem is domain-specific. [AGSS86] and [AH91] showed that, unlike the safety problem, the state-safety problem is decidable for many domains.

It had remained unknown whether the state-safety problem is decidable for each domain with decidable FO theory.

Although the set of safe formulas is unsolvable (and not even enumerable), it may be possible to impose certain syntactical restrictions on the class of formulas that we are going to use such that the safe *queries* are exactly those ones which can be formulated with these restrictions.

In other words, the problem can be put as follows. Does there exist a recursive subclass of safe formulas such that every safe formula is equivalent to one in this subclass?

We will call such a subclass a *recursive syntax.* One may consider recursively enumerable (r.e.) subclasses as well. As usual, for this kind of problems, the existence of an r.e. syntax implies the existence of a recursive syntax, so we are going to henceforth concentrate on the existence of a recursive syntax.

This approach is due to M.Y. Vardi [Var81] and J.D. Ullman [Ull82], and since then it has been developed in many publications, including [Ull88], [Van91], and [Hir91].

J.D. Ullman in [Ull82] (and somewhat more clearly in [Ull88]) shows that a recursive syntax for domain-independent queries does exist. A. Van Gelder and R. Topor [Van91] address the issue of efficiency of syntax.

For some primitive domains, for instance, for the domain with only the equality predicate, the classes of finite and domain-independent queries coincide, so the syntax actually work for both the classes. For some more developed domains, these classes differ, however, it is not hard to develop a syntax for finite queries for most of the domains considered in the literature.

This syntactical approach has definite advantages over the state-safety one, especially as in more and more cases the actual queries to databases are formulated by software rather than people. Thus, naturality of query languages becomes perhaps less important, while stability becomes more important.

On the other hand, it may be that the unsafety of a formula is due to a rather rare situation, and then it may be useful to be able to use this formula for as long as the actual state-unsafety does not happen.

Again, it had remained unknown whether a recursive syntax for finite formulas exists for every domain.

In [ST95a] it is shown that there exists a recursive domain with decidable theory in which (1) there is no recursive syntax for finite queries, and in which (2) the state-safety problem is undecidable.

1.4. **Ordered domains and generic queries.** For example, consider the ordered set of rational numbers. A FO query, then, is a mapping that maps every finitely representable database state into a new finitely representable relation. Observe that, if we take any such pair — an f.r. database state and the f.r. state which is the answer to the query in this state — and uniformly change some of the constants used in the finite representations while preserving order between different constants, then the new pair agrees with the mapping. In this sense, all the queries that can be expressed in the first order language FO are *generic.*

There are some rather simple generic queries, however, that are not expressible. For example, the Boolean query that says that the cardinality of a finite set — a unary relation — is even is not FO expressible. More examples can be found in [KKR90, KKR95, KG94]. The problem we are interested in is to try to increase the expressive power of FO, while preserving the genericity.

Let us consider an arbitrary ordered domain. The original notion of generic query [CH80] referred to the =-generic queries over finite database states, that is, the queries (over finite states) which are preserved under arbitrary (not necessarily $<$-preserving) permutations of the domain. Some practically interesting queries, say, graph properties, are indeed =-generic.

The first order queries expressible without using the order relation (the pure FO queries) over the domain are *generic* (see [CH80]), meaning that they are preserved under arbitrary permutations of the domain.

The expressive power of the pure FO with respect to generic queries is, however, severely limited — a classical example is inexpressibility of the parity query asserting that the cardinality of a finite relation in the database scheme is even. One of the ways to try to enhance the expressive power of the query language is by allowing certain *domain functions/relations,* or *givens* to be used in the queries. The simplest example is the relation $<$ of linear order. These givens are considered to be a part of the domain — rather than of the database scheme — and to have a fixed meaning. Throwing in such givens does obviously increase the expressive power of FO, but what is often not obvious is whether any new *generic* queries become expressible.

Yu. Gurevich [Gur90] showed that there are =-generic queries that are FO expressible with $<$ over finite states, but not without $<$. Here is a version of his example.

Let K be the class of all finite Boolean algebras with an even number of atoms. This class cannot be axiomatized within the class of finite structures by a first order sentence because otherwise, by compactness, there would exist infinite atomic Boolean algebras B and B' such that the sentence holds in B but fails in B', in contradiction with the completeness of the theory of infinite atomic Boolean algebras (see e.g. [CK90] for the latter fact). However, $K_<$, the class of expansions of algebras in K by linear orders, is axiomatizable in the class of finite structures by a first order sentence ψ which is the conjunction of the axioms for Boolean algebras, the axioms for linear orderings, and a sentence expressing that there is an element containing exactly atoms at an even position (in the ordering induced on the atoms) and containing the last atom. It follows that, over any infinite ordered universe U, the FO query obviously corresponding to ψ is not equivalent to a pure FO query for finite states, even though it is $=$-generic.

Note that although the language FO($<$) of first order logic with a relation of linear order does indeed express more generic queries than the pure FO, the parity continues to be inexpressible.

Naturally, we may ask whether, over a certain domain, it is possible to express even more $=$-generic queries using extended signatures. We observe however that, because each $=$-generic query being $<$-generic, the collapse results like the ones established in this paper are automatically transferred to the case of $=$-genericity.

So the natural question has been, whether allowing certain other givens, in specific situations, enhances the expressive power of generic FO($<$) even more. And while in some situations the answer is trivially affirmative — for example, allowing $+$ and $\times$ over integer (or rational) numbers makes it possible to express all computable queries — in others the question may be hard.

Let $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, and $\mathbb{R}$ be the sets of all natural, integer, rational, and real numbers, respectively. Practically, the most interesting cases have been:

- $(\mathbb{Q}, <, +)$ and $(\mathbb{R}, <, +)$
- $(\mathbb{Z}, <, +)$ and $(\mathbb{N}, <, +)$
- $(\mathbb{R}, <, +, \times)$

For example, in the papers of Grumbach et al. [GST95, GS95] extended queries over $(\mathbb{Q}, <, +)$ were considered. Clearly, $x + x = x$ defines 0, which is not definable without $+$; on the other hand, queries like $x + x = x$ are not about order at all, as they are not preserved under order automorphisms. To level the playing field, they consider *locally generic* extended queries only, that is the queries preserved under partial $<$-isomorphisms of the universe. To put it another way, while arithmetical computations can be carried out inside the query, the input-output relation defined by this query may rely only on the order properties of its input.

Over rational numbers, the FO queries that only use $<$ were shown to have the uniform data complexity AC^0 [KG94]. Attempts to distinguish the resulting extended queries from order queries in this domain using specific combinatorial or spatial queries not in AC^0 — like parity, Eulerian traversal, or region connectivity — have been unsuccessful, and, finally, [GST95, GS95] proved the AC^0 uniform data complexity for the extended queries over finitely representable inputs *with integer constants only.* However, the question of whether or not extended queries are more expressive than order queries, has remained open, and as Grumbach and Su [GS94] pointed out, " ... there is a serious lack of proof techniques ... " in this area.

This very problem can be considered in a general form. Consider a list Ω of relational names. Consider two signatures $L_0 = \{<\}$, and $L = \{<\} \bigcup \Omega$. For a database scheme $SC = \{R_1, \ldots, R_n\}$, denote $L_0^+ = L_0 \bigcup SC$ and $L^+ = L \bigcup SC$. Call the first order language in L_0^+ *restricted,* and the first order language in L^+ *extended.* The *general problem* is then, under which conditions on the domain are generic extended queries reducible to restricted queries?

Notice that this problem admits several interpretations:

- "generic" may be understood as "locally generic", or a different notion of genericity may be used; for instance, preservation under order automorphisms is simply referred to in this paper as "genericity". While for many domains (for the rational numbers with $+$, for the real numbers with $+, \times$, and, generally, for an arbitrary doubly-transitive domain) it does not make a difference, for some other domains it may
- "reducible" may mean that the two languages are equivalent, or that a recursive translation exists
- the problem may be formulated for either all, finite or infinite, database states, or for f.r. states only, or for finite states only

In [ST96, BST97a, BST97b, BST96] it was shown that, if all possible states were considered no translation would be possible even in such simple an example as the additive group of rational numbers.

But of course the really interesting cases are those of finite and of finitely representable database states. In [ST96, BST97a, BST97b, BST96] it was shown that these two cases can be treated uniformly. One of the main results of the papers is that, over every ordered domain, finitely representable states can be uniformly represented as finite states of another database scheme, with the additional property that these finite database states are FO expressible (in the restricted language) in the old database scheme, and vice versa. This technique, in effect, allows us to lift any result on translatability of extended queries into restricted queries over *finite database states*, to the *finitely representable* states. The recursiveness of translation is preserved as well.

This technique can also be used to expand applicability of several other results for finite database states to the case of finitely representable states.

1.5. **Recursive translation.** Paradaens et al. [PVV95] considered real numbers with $+$, and showed that, over finite database states, generic extended queries can be recursively[2] translated into restricted (pure order) queries. Due to our lifting result, the same is automatically true for all finitely representable states.

In [ST96], Stolboushkin and Taitslin proved a more general result on recursive translation of generic extended into restricted FO queries over an arbitrary ordered divisible Abelian group, thus answering, for example, the question from [GST95, GS95] of the role of addition in databases over rational numbers (in the non-effective sense, this problem was independently solved in [BDLW96]).

Although recursive, this translation is highly inefficient computationally. The size of the pure order formula generated by our algorithm is multi-exponential in the size of the original generic formula with addition. We feel this is a natural phenomenon, in the sense that no efficient (say, polynomial-size) translation is possible. This offers yet another angle of looking at the relative expressive power of extended and restricted queries. Indeed, using an extended language may be beneficial in that it may offer a compact way of expressing generic queries.

Notice also, the set of generic extended queries is undecidable. Our translation algorithm *assumes* genericity, and, if applied to a non-generic query, gives a non-equivalent formula of the restricted language.

1.6. **Non-recursive translation.** Benedikt et al. [BDLW96] showed that the generic extended, and restricted queries have the same expressive power over every *o-minimal* domain,[3] the notion introduced in [PS86, KPS86, PS88]. Examples of o-minimal structures include, for instance, the real numbers with $+$, $\times$, the exponentiation and $<$, as well as many other structures. Since every ordered divisible Abelian group is o-minimal, this, in one sense, covers the results discussed in Subsection 1.5.

Notice, however, that in another sense, this result is of a different nature. First, the proof in [BDLW96] is not constructive and does not give an algorithm for translation. Further, this proof cannot be made constructive. Indeed, take an o-minimal structure whose first order theory is undecidable, while the first order theory of $<$ alone is decidable, for example, the structure $(\mathbb{R}, +, \times, <, c)$, where c is a non-computable real number. If a recursive translation existed, this would lead to a contradiction.

In [BST97a, BST97b, BST96], we suggested an approach that gives substantially stronger results of this sort. The approach is based on the observation that the expressibility of a locally order-generic extended query over finite states over a universe as a restricted query is a property of the complete first order theory of the universe rather than the universe itself. Therefore

[2] Although the algorithm is not explicit in their paper.

[3] Again, they considered finite states only, but this can be lifted to finitely representable states using our technique in this paper.

we can use the known model-theoretic technique of saturated models to study this property of the theory of the universe.

Firstly, we give a necessary and sufficient condition for an extended query to be equivalent to a restricted query.

Secondly, this technique is developed especially for locally generic queries.

Thirdly, we formulate a very general condition on the domain — the so-called *Pseudo-finite Homogeneity Property* — that ensures collapse of locally order-generic FO queries over this domain to pure order queries. However, proving the Pseudo-finite Homogeneity Property for a specific domain may be a bit technical. We introduce a condition on the domain, the Isolation Property, which ensures the Pseudo-finite Homogeneity Property.

Fourthly, we identify a broad class of domains — the so-called *quasi-o-minimal domains* — which all satisfy the Isolation Property. Examples of the quasi o-minimal domains include the following:

- all o-minimal domains
- the integer or natural numbers with $+, <$
- the ordered set of real numbers with the distinguished subset of rational numbers
- ordered unions of o-minimal domains

The Isolation Property is broader than the quasi-o-minimality: for example, every structure of the form $(A, <, E)$, where $<$ is a dense linear ordering on the set A, and E is an equivalense relation on A with two dense classes, satisfies the Isolation Property but is not quasi-o-minimal.

The Pseudo-finite Homogeneity Property is broader than the Isolation Property: we prove that for the structure $(\mathbb{R}, +, <, \mathbb{Q})$ the Pseudo-finite Homogeneity Property holds but the Isolation Property fails. In particular, for this structure every generic extended query over finite states is equivalent to a restricted query. This immediately implies the analogous collapse result for any structure of the form $(A, <, E)$, where $(A, <)$ is a dense linearly ordered set without endpoints, and E is an equivalence relation on A with infinitely many classes all of which are dense.

The general setting we consider really gives some other concrete examples of collapse results. For instanse, the collapse result holds for any structure of the form $(\mathbb{R}, +, <, F, f_\alpha)_{\alpha \in F}$, where F is a subfield of $\mathbb{R}$, and f_α is a name for the unary operation of multiplication by the scalar α. However, it is easy to see that for the structure $(\mathbb{R}, +, \times, <, \mathbb{Q})$ the collapse result fails. So we are really standing near a boundary.

1.7. **Safe** $Datalog^{\neg, <_z}$**-programs.** Although the domain $\mathbb{Z}$ does not admit elimination of quantifiers, its definitional expansion by the so-called *gap-orders* $<_g$ for all natural numbers g already admits elimination of quantifiers. $x <_g y$ means $x + g < y$.

Thus, this expanded domain admits effective *bottom-up evaluation* of first-order queries in *closed form* with respect to f.r. states. "The closed form" means that whenever you start from an f.r. state, you end up having an f.r. answer that can therefore be stored in the database and used in future queries as an extensional predicate. "Bottom-up evaluation" refers to the process of evaluating queries according to their structure, from inside-out, by constructing for each sub-formula a finite representation of its value. This process is much more efficient than the tuple-based evaluation.

However, the expressive power of first-order queries in this domain is severely limited. This motivated research into using constraint logic programs (see [JL87, JM94]) for querying finitely representable databases over the integer order. Logic programs without negation, when they terminate, result in f.r. answers too. This means that the result of one program, or its negation, can be used as input for another program. This leads to the notion of Datalog with *stratified negation,* or $Datalog^{\neg,<_z}$, where negations are allowed, but only w.r.t. the intensional predicates whose computation already terminated (cf. [CH82]).

This machinery only works well, however, if the Datalog program terminates. If it does not, the construction collapses. One remedy is to consider only those $Datalog^{\neg,<_z}$-programs whose termination is guaranteed for all inputs. Such programs often are called *safe.* Notice that this definition is semantical in nature.[4]

Revesz [Rev95] introduced a syntactical notion of safety for $Datalog^{\neg,<_z}$-programs, which guarantees semantical safety. The syntax is remarkably powerful — queries expressible in this syntax may have non-elementary complexity — and yet easy (=PTime) to check. As a matter of fact, it was not clear what kind of (semantically) safe queries, if any, could not be expressed in this syntax.

In [ST95c], this problem was ultimately settled by showing that no syntax exists for *all* semantically safe queries of $Datalog^{\neg,<_z}$. In particular, the syntax introduced in [Rev95] is incomplete. Formally, it was shown that any recursively enumerable set of $Datalog^{\neg,<_z}$-programs either contains infinitely many unsafe programs, or does not contain any program for infinitely many safe $Datalog^{\neg,<_z}$-definable queries.

Of course, the result implies undecidability of safety for $Datalog^{\neg,<_z}$ as a whole, i.e., that one cannot decide for a program R whether it is safe. However, the result hits deeper in that we show impossibility of *any* syntactical safety restriction on the $Datalog^{\neg,<_z}$-programs that would not simultaneously be semantical. As a matter of fact, oftener than not an effective syntax for an undecidable class of programs does exist. By way of example, consider the ever popular class PTIME. Again, one cannot generally say whether a given program is in PTIME. However, it is easy to come up with a syntactical class of programs that consists of PTIME programs and covers the whole class PTIME function-wise.[5]

[4] To be sure, the notion of safety only makes sense when a specific query evaluation algorithm is fixed. Within this paper, we concentrate on the bottom-up evaluation algorithm.

[5] Say, take only the programs that track their own execution time and terminate when a target polynomial is reached.

On the technical side, one of the main results of [ST95c] is that, under the bottom-up semantics, for any Turing machine one can effectively construct a $Datalog^{\neg,<_z}$-program that computes the same function and is safe whenever the machine is total. Although by appearance, the result looks similar to the one by P. Revesz (Proposition 2.3 in [Rev93]) that any Turing-computable function is *expressible* by a query of $Datalog^{\neg,<_z}$, a closer look reveals that the the two results are altogether different. To emphasize only one distinction, the programs that *express* (total) Turing-computable functions in [Rev93] need not terminate under the bottom-up semantics, hence, they may not be safe.

1.8. **Notation.** Notation is usual. ∎ denotes either the end of the proof or that the proof of the statement is omitted.

2. A recursive domain with decidable FO theory which has no recursive syntax for safe queries

The goal of this section is to present a recursive domain with decidable FO theory where there is no recursive syntax for finite queries, and where the state-safety problem is undecidable.

In quest for a decidable theory with this property the authors thoroughly reviewed many theories considered in [Rab77] and [ELTT65], however, were unable to find an appropriate theory.

In [ST95a], it was proposed a new domain specially coined to serve the needs. The naturality of this domain can be further argued upon.

A reader with a background in Recursion Theory may notice in our design certain similarities to Kleene's predicate. However, throwing in the full Kleene's predicate would make the theory undecidable. So what we actually are doing, is we are weakening Kleene's predicate to the extent that the first-order theory becomes decidable.

Now finiteness of a query implicitly involves a second-order property, and we manage to use this peculiar second-order property in conjunction with our predicate to express totality of recursive functions.

So much for the underlying informal ideas.

Now, formally, let us define the domain as follows.

The domain is the set of all possible words (or strings) in the alphabet $\{1, *, \#, -\}$. The signature contains the only ternary predicate symbol P, as well as all the constant words in the alphabet. Also, we consider the language with equality $=$.

Let us consider the standard single-tape Turing machines in the alphabet $\{1, -\}$. As usual, the machines use $-$ as a white-space marker. Initially, an input word $w \in \{1, -\}^*$ written on the tape is surrounded by infinitely many $-$ from both sides, and machines always start from reading the leftmost character of the word w. Internally, Turing machines use the two-character alphabet $\{1, -\}$ and throughout the computation, modify characters written on the tape. A machine can run forever, but if it stops, it leaves a finite word written on the tape. If at this moment the tape only contains $-$ in all positions, the result of the computation is defined as the empty word ϵ. Otherwise, the result is the leftmost word in the alphabet $\{1\}$ written on the tape and surrounded by $-$. A Turing machine is called total iff it stops for any input.

The Turing machines themselves can be represented as strings in the alphabet $\{1, -, \#\}$ with # being a delimiter (we require that every machine contains at least one #). The details of a particular representation are not otherwise important.

Let a word $w \in \{1, -\}^*$ and a Turing machine $M \in \{1, -, \#\}^+$ be given. We now define a *trace* of M in w as a sequence of "snapshots" of a partial computation of M in w separated with $*$. A trace starts with $M*$ and then, for each step in the computation, contains the internal state of the machine M, the tape (the minimal part of it that covers all non-$-$ characters), and the position of the head in the tape, all separated with $*$. For instance, the first snapshot always is $1 * w *$.

Thus, if M does not stop in w, there are infinitely many different traces of M in w. However, if it does stop in w, then the number of different traces is finite.

A trace therefore is a word in the alphabet $\{1, *, \#, -\}$.

Note that the machines, the input words, and the traces, being all written in different alphabets, do not intersect. Also, there are words that are of neither of these three types, which we will call "other words".

Now the only (ternary) signature predicate P is defined as follows.

$P(M, w, p)$ iff M, w, p are a machine, an input word, and a trace, respectively, and p is a trace of M in w. The equality is also allowed. Let us denote this domain $\mathbb{T}$.

By the *Theory of Traces* we mean the set of true first-order pure domain sentences of $\mathbb{T}$.

Theorem 2.1 ([ST95a]). *The Theory of Traces is decidable.*

∎

Theorem 2.2. *The set of finite formulas of the domain $\mathbb{T}$ does not have an effective syntax.*

Proof. Consider a database scheme that consists of one constant symbol[6] c.

Given a Turing machine M, consider the following formula $\psi_M(x)$:

$$P(M, c, x).$$

Observe that the formula $\psi_M(x)$ is finite iff M is total. Indeed, if M is total, then, for any c, only finitely many x's may satisfy $P(M, c, x)$.

If, on the other hand, M is not total, there exists c such that infinitely many traces x satisfy $P(M, c, x)$. Then, obviously, $\psi_M(x)$ is infinite.

[6]Of course this formally is not a database scheme, but this technicality will be taken care of later.

Now suppose that the theorem does not hold. Then there exists a recursive enumeration $\phi_1(x), \phi_2(x), \ldots$ of finite formulas (that use c in addition to the domain constants and to the predicate P) with one free variable such that any finite formula with one free variable is equivalent to one in this list.

Without loss of generality, we may assume that a variable, say z, is not used in the formulas of this list.

Consider a recursive enumeration of all, total or not, Turing machines, $M_1, M_2, \ldots$.

Given a machine M_k and a formula $\phi_r(x)$, consider the formula

$$(\forall z)(\forall x)\left(\psi_{M_k}(x)\left[\frac{z}{c}\right] \longleftrightarrow \phi_r(x)\left[\frac{z}{c}\right]\right),$$

where $\left[\frac{z}{c}\right]$ is the operation of substituting the variable z for the constant symbol c in a formula.

This last formula is therefore a pure domain formula, and because the decidability of the theory, we can check whether it is true or not.

Now if it happens to be true, we know, that M_k is a total machine, because the truth of this sentence implies that $\psi_{M_k}(x)$ is finite.

On the other hand, if M_k is total, then ψ_{M_k} is finite, and, therefore, for some r, the above sentence is going to be true, for $\phi_1(x), \phi_2(x), \ldots$ include all finite queries with one variable.

Hence, by continuously analyzing all pairs of k and r, we can establish a recursive enumeration of *all* total Turing machines.

But this is known to be impossible. A straightforward proof of this fact can be obtained by a simple diagonalization.

Hence, a contradiction.

Finally, notice that we do not need to stick with the constant c. A database scheme may contain, say, one unary relation R instead of the constant symbol, and then we will define the totality formula $\psi_M(x)$ as follows:

$$(\forall x, y)(R(x) \wedge R(y) \implies x = y) \wedge (\exists y)(R(y) \wedge P(M, y, x)).$$

The same proof can be carried out here with minor adjustments. ∎

Corollary 2.3. *For no extension $\mathbb{T}'$ of $\mathbb{T}$ that has a decidable theory, a recursive syntax exists for finite queries.*

Proof. The proof of Theorem 2.2 continues to work. ∎

The situation is no better with the state-safety:

Theorem 2.4. *The state-safety problem is undecidable for $\mathbb{T}$.*

Proof. In the notation of Theorem 2.2, notice that $\psi_M(x)$ is finite in the state c iff M stops starting from the value of c. While it is undecidable to determine whether a Turing machine stops in an input. ∎

3. Locally order-generic queries

In the section we represent results from [BST96]. Omitted proofs can be found in [BST96]. The paper is available as ps-file for anybody using ftp address `ftp.tversu.ac.ru` (directory `/pub/Taitslin`, file `last2.ps`).

3.1. **Preliminaries.** A structure of a relational signature L is a non-empty set with a mapping that assigns to every relational symbol in L a relation of the same arity over the set. Let U be an infinite structure over the signature L. This structure is called the *universe.* In this section, we always consider ordered universes. This means that L includes a binary relational symbol $<$ whose interpretation in U satisfies the axioms of linear order. Let us denote $L_0 = \{<\}$, and $\Omega = L \setminus L_0$.

Databases operating over U use non-signature relational symbols as well. A *database scheme* SC is a finite collection of relational symbols of fixed arities. A *database state* (over U) for the database scheme is an assignment to these relational symbols of concrete relations of corresponding arities over U. These relations are called *database relations.* A database state is called a *finite database state* if all the relations are finite. The set of all elements of the universe that occur in some tuple in some relation of a database state s is called the active domain of s; we denote it by $ad(s)$. We denote $AD(x)$ a first order formula of the signature SC which says that x is an element of the active domain. It means that for a database state s and $a \in U$, $AD(x)$ is true in (U, s, a) iff $a \in ad(s)$.

We fix a database scheme SC and denote $L_0^+ = L_0 \bigcup SC$, and $L^+ = L \bigcup SC$.

A *database query* can formally be defined as a mapping that takes in a database state (of a fixed database scheme), and produces a new relation, of a fixed arity, over U. Thus, every query has an arity. Specifically, queries of arity 0 are called *Boolean queries.* A Boolean query defines a mapping from database states to $\{0, 1\}$, or, in other words, subsets of all possible database states of a given database scheme.

Queries can be formulated using *query languages,* the simplest being the language of first-order logic FO. Formulas (queries) of this language use $=$, as well as the relational symbols of the signature and of the database scheme. Thus, a database state essentially defines a structure of a larger signature with U as the domain; then a formula with n free variables defines an n-ary relation over U.

Generally, an FO query may yield an infinite answer even in a finite database state. [KKR90] introduced the notion of *finitely representable database state* as a database state where every relation corresponding to a relation name from SC is defined (independently of the others) by a quantifier-free formula using $=, <$, and constants for the elements of U. The formula is a *finite representation* of the relation.

We consider two languages for querying. Queries of the first one are FO formulas of the signature L_0^+ — we call them *restricted.* Queries of the second language are FO formulas of the signature L^+ — we call them *extended.*

A query is said to be *generic,* iff they are preserved under order-preserving permutations of U.[7] It is easy to see that the restricted queries are generic. In other words, if $\phi : U \to U$ is an automorphism of $\langle U, <\rangle$, and a restricted query Q transforms a database state s into a relation R, then Q transforms $\phi(s)$ into $\phi(R)$; in other words, $Q(\phi(s)) = \phi(Q(s))$. The problem with extended queries is, they may be not generic.

We will also use a stronger notion of *locally generic* query. A k-ary query Q is said to be *locally generic over finite states* if $\bar{a} \in Q(s)$ iff $\phi(\bar{a}) \in Q(\phi(s))$, for any partial $<$-isomorphism $\phi : X \to U$ with $X \subseteq U$, for any finite state s over X, and for any k-tuple $\bar{a}$ in X.

For any finite representation σ over a subset X of U and for any partial $<$-isomorphism $\phi : X \to U$, a finite representation $\phi(\sigma)$ can be naturally defined, by replacing any parameter a that occurs in σ with the parameter $\phi(a)$. So, for finitely representable states, the notion of local genericity can be defined as follows. A k-ary query Q is said to be *locally generic over finitely representable states* if $\bar{a} \in Q(\sigma)$ iff $\phi(\bar{a}) \in Q(\phi(\sigma))$, for any partial $<$-isomorphism $\phi : X \to U$ with $X \subseteq U$, for any finite representation σ over X, and for any k-tuple $\bar{a}$ in X. Here we denote by $Q(\sigma)$ the state into which the query Q transforms the state finitely represented by σ.

Since a finite n-ary relation $\{\bar{a}_1, \dots, \bar{a}_n\}$, where $\bar{a}_i = (a_{i1}, \dots, a_{in})$, can be finitely represented by the formula $\bigvee_{i=1}^{m} \bigwedge_{j=1}^{n} x_j = a_{ij}$, every query which is locally generic over finitely representable states is locally generic over finite states, too. On the other hand, a query which is locally generic even over *all* states can be not locally generic over finitely representable states: an example is the Boolean query '$P \neq \emptyset$'; it is obviously locally generic over all states over $\mathbb{Z}$, but $0 < x < 1$ defines in $\mathbb{Z}$ the empty set, even though the set defined by $0 < x < 2$ in $\mathbb{Z}$ is not empty.

Of course, every locally generic query is generic. Conversely, for some domains, every generic query is locally generic. A sufficient condition for this is the so-called *double transitivity* of the domain: a domain is called doubly transitive if for any $a_1 < b_1$ and $a_2 < b_2$ in the domain, there exists a $<$-automorphism of the domain mapping a_1 to a_2, and b_1 to b_2. For instance, the real and rational numbers are doubly transitive, while the integer numbers are not. The Boolean query *'there are even and odd numbers in P'* is an example of a query which is generic but not locally generic over finite states over $\mathbb{Z}$. Moreover, even a restricted query can be not locally generic: for example, the restricted Boolean query *'P is convex'* is not locally generic over finite states over $\mathbb{Z}$.

3.2. **Impossibility of translation over arbitrary states .** The goal of this subsection is to compare restricted and generic extended queries from

[7]As discussed in Introduction, the term "generic" is sometimes understood in a more restrictive sense.

the viewpoint of their expressive power over *all possible states,* whether finitely representable or not. We show that, in general, extended generic querying is more expressive than restricted, even in very simple situations.

Theorem 3.1. *There is an extended Boolean query Q over $(\mathbb{Z}, +, <)$, the ordered group of integer numbers, such that*

(1) *Q is generic over all database states; in particular, Q is generic over all finite states;*
(2) *Q is not equivalent, over finite database states, to a restricted query; in particular, Q is not equivalent, over all database states, to a restricted query.*

Proof. Let Q be the query *'there are even and odd numbers in P'*. The query Q is generic over all possible database states.

Clearly, there is an extended FO query that expresses Q, for all possible database states.

But Q cannot be expressed as a FO restricted query, for finite database states. ∎

Note that Q constructed in Theorem 3.1 is obviously not locally generic. Moreover, it will be shown that *every* FO extended query, which is locally generic over finite states over $(\mathbb{Z}, <, +)$, is equivalent, for finite database states, to an FO restricted query. A similar result will be proved for $(\mathbb{Q}, <, +)$, the ordered group of rational numbers.

By the way, the mentioned result concerning $\mathbb{Z}$ has a curious corollary: *the query '$|P|$ is even' cannot be expressed as an extended FO query for finite database states over $(\mathbb{Z}, <, +)$, as opposed to the query 'there are even and odd numbers in P'.* Indeed, the query '$|P|$ *is even*' is obviously locally generic, even over all database states, and essentially the same arguments, as in the proof of Theorem 3.1, show that the query is not equivalent, for finite database states, to an FO extended query over $(\mathbb{Z}, <, +)$. Note, by contrast, that the query '$|P|$ *is finite*' can be expressed as a restricted FO query over $(\mathbb{Z}, <, +)$, because a set of integers is finite iff it is bounded.

It is natural to ask whether Theorem 3.1 holds for $\mathbb{Q}$ instead of $\mathbb{Z}$. In this situation, in contrast to the case of $(\mathbb{Z}, <, +)$, the notions of genericity and local genericity coincide. However, for $(\mathbb{Q}, <, +)$, we will give an example of an extended query which is generic over all database states, but not equivalent, over all database states, to a restricted one. That example draws a line between finite and finitely representable database states, on one side, and essentially infinite states, on the other.

In fact, we will prove a more general result:

Theorem 3.2. *The extended querying is more expressive than the restricted one with respect to generic Boolean queries over any divisible Archimedean ordered Abelian group not isomorphic to the ordered group of reals.*

Classical examples of divisible Archimedean ordered Abelian groups are the ordered groups of rational and real numbers. It is known that, up to isomorphism, Archimedean ordered groups are exactly the subgroups of the ordered group of reals. The following is an example of a uncountable divisible Archimedean ordered Abelian group not isomorphic to the ordered group of reals.

Let b be an irrational number. Consider a basis B of $\mathbb{R}$ over $\mathbb{Q}$, containing b; clearly B is of power of continuum. Let G be a $\mathbb{Q}$-subspace of $\mathbb{R}$ generated by $B \setminus \{b\}$. Then G is a required group. Indeed, clearly it is Archimedian and divisible. The orders on G and $\mathbb{R}$ are not isomorphic: the order on G is not complete as b is not in G.

We don't know whether the result of Theorem 3.2 holds for the ordered group of reals.

To prove Theorem 3.2, it suffices to prove the following two results.

Theorem 3.3. *Let $(A, +, <)$ be an Archimedean ordered Abelian group. Then the finiteness of database states over A is expressible by an extended FO query.*

∎

Theorem 3.4. *Let A be a set of reals containing $\mathbb{Q}$. Then the finiteness of database states over A is expressible by a restricted FO query iff $A = \mathbb{R}$.*

∎

Note, by contrast, that in $(\mathbb{Z}, <)$ the finiteness is expressible by a restricted FO query because a set of integers is finite iff it is bounded.

In the special case when A is countable, Theorem 3.4 admits an especially simple proof. If φ expressed the finiteness of R in $(A, <, R)$ then, by compactness and the Löwenheim-Skolem theorem, there would be an infinite subset in a countable dense ordered set without endpoints, for which φ holds. As every countable dense ordered set without endpoints is isomorphic to $(A, <)$, we would have a contradiction with the choice of φ.

3.3. **Canonical representation of finitely representable relations .** The goal of this subsection is to show that finite and finitely representable states can be treated uniformly. To achieve this goal, we show that finitely representable relations can be represented uniformly by finite relations, of a different signature, such that these f.r. states and their finite "codes" can be mapped to each other by restricted queries.

We begin with a simple example. Consider the following finitely represented unary relation P_0 on $\mathbb{Q}$:

$$(1 < x \leqslant 2) \vee (3 \leqslant x < 4) \vee (5 < x).$$

To reconstruct P_0, it suffices to know:

- the set of constants B_0 used in the representation — in our case it is $\{1, 2, 3, 4, 5\}$, and, moreover,

- which of the singletons and the open intervals

$$(-\infty, 1),\quad \{1\},\quad (1,2),\quad \{2\},\quad (2,3),\quad \{3\},\quad (3,4),\quad \{4\},\quad (4,5),\quad (5,\infty)$$

  are contained in P_0.

The latter sets can be characterized as minimal open intervals and singletons which can be defined using constants from B_0; we will call them B_0-minimal 1-cells. Clearly, P_0 is the union of all B_0-minimal 1-cells which are contained in P_0.

The set of constants over which P_0 can be defined is not uniquely determined by P_0; for example, P_0 can be represented over the set $\{0,1,2,3,4,5\}$ by the formula

$$((0 < x \leqslant 2) \vee (3 \leqslant x < 4) \vee (5 < x)) \wedge (1 < x).$$

However, the constant 0 is oviously irrelevant here as it is shadowed by the second conjunct. In fact, the constants which are really relevant are just the boundary points of P_0. Clearly, there is a unary restricted FO query δ which, for any subset P of $\mathbb{Q}$ as an input, yields its boundary as an answer.

We show that the information which is contained in the second item can be obtained from P_0 by means of several FO restricted queries (which can be uniformly applied to any finitely represented subset of $\mathbb{Q}$).

Let B be a boundary set of such a P; it is a finite set. There are 5 types of B-minimal 1-cells:

$$(0)\ \mathbb{Q};\qquad (1)\ \{b\};\qquad (2)\ (-\infty, b);\qquad (3)\ (b,\infty);\qquad (4)\ (b,b');$$

here $b, b' \in B$, and in all the cases except (1) the 1-cells do not contain any point from B.

Consider the following 5 relations S_i on B ($i < 5$). The relation S_0 is 0-ary; the relations S_1, S_2 and S_3 are unary, and the relation S_4 is binary. We put

- $S_0 = \mathbf{true}$ iff $\mathbb{Q} = P$
- $S_1(b)$ holds iff $b \in B$
- $S_2(b)$ holds iff b is the least element of B and P contains $(-\infty, b)$
- $S_3(b)$ holds iff b is the greatest element of B and P contains (b, ∞)
- $S_4(b, b')$ holds iff b, b' are subsequent elements of B, and P contains the interval (b, b')

In other words, S_i contains the information which of the B-minimal 1-cells of type (i) are contained in P.

It is easy to write down a FO query σ_i which, for every finitely represented subset P of $\mathbb{Q}$ as an input, yields the finite relation S_i as the answer. Clearly, P can be recovered from the finite relations B and S_i's by means of a FO query π, because P is the union of all B-minimal 1-cells which are contained in P.

So, we have shown that, for any finitely representable subset P of Q, we can find, uniformly in P by means of FO queries, a finite collection of finite relations on $\mathbb{Q}$, from which P can be uniformly recovered by means of a FO query.

Our goal is to prove an analogous result for finitely represented relations of arbitrary arity. Here the idea is essentially the same, but some new important points appears. We illustrate this for the case of arity 2.

Consider a binary finitely represented relation P on $\mathbb{Q}$. We may assume that P is contained in one of the three relations

$$\{(a,b) : a < b\}, \quad \{(a,b) : a > b\}, \quad \{(a,b) : a = b\},$$

because P is the disjoint union of the intersections of P with these three relations, and the intersections are finitely representable. Assume, for example, that $P \subseteq \{(a,b) : a < b\} = D$.

It can be proven (it is not obvious!) that among the finite subsets of $\mathbb{Q}$, over which P can be finitely represented, there is a least one; we call it the set of boundary points for P and denote it by ∂P. For example, if P is the relation

$$(0 < x < y < 1) \vee (1 < x < y < 2)$$

then $\partial P = \{0, 1, 2\}$. Moreover, it turns out that ∂P can be obtained from P by means of a FO query, uniformly in P.

For a finite subset B of $\mathbb{Q}$, we define a simple binary relation on $\mathbb{Q}$ over B to be a finite union of "rectangulars" defined over B; that is, the simple binary relations over B are those which can be finitely represented by disjunctions of conjunctions of formulas of the forms

$$x = b, \quad x < b, \quad x > b, \quad y = b, \quad y < b, \quad y > b,$$

where the parameters b are taken from B. It is easy to show that simple binary relations on $\mathbb{Q}$ can be characterized as the binary relations that are invariant under all order automorphisms of $\mathbb{Q}$ which pointwise stabilize B.

We will show that there is a least simple binary relation on $\mathbb{Q}$ over ∂P containing P; we denote it by $\mathrm{Inv}(P)$. For the P from the example above, $\mathrm{Inv}(P)$ is the union of two squares, $(0,1) \times (0,1)$ and $(1,2) \times (1,2)$. It is easy to see that $\mathrm{Inv}(P)$ can be uniformly obtained from P by means of a binary first order query. It can be shown — it is the crucial point — that it always the case that $P = \mathrm{Inv}(P) \cap D$.

We call a set of the form $I \times J$, where I and J are B-minimal 1-cells, a B-minimal 2-cell. It is easy to see that any simple binary relation over B can be uniquely decomposed into a disjoint union of B-minimal 2-cells. For instance, in the example above $\mathrm{Inv}(P)$ is the disjoint union of two $\{0,1,2\}$-cells $(0,1) \times (0,1)$ and $(1,2) \times (1,2)$. Clearly, there are finitely many B-minimal 2-cells, for any finite B.

If we know ∂P, to reconstruct $\mathrm{Inv}(P)$, we need only to know which of the ∂P-minimal 2-cells are contained in $\mathrm{Inv}(P)$.

As there are 5 types of ∂P-minimal 1-cells, there are 25 types of ∂P-minimal 2-cells. With every type $\tau \in 5^2$ of ∂P-minimal 2-cells we associate a relation S_τ on the finite set ∂P with the information which of the ∂P-minimal 2-cells are contained in $\mathrm{Inv}(P)$. The arity of S_τ depends on τ and is equal to the number of the b's involved in representations of 2-cells of type τ.

Say, with the 2-cells $I \times J$, where I is of type (1) and J is of type (4), we associate the following ternary relation S_{14} on ∂P. For $a, b, c \in \partial P$, we consider $S_{14}(a, b, c)$ to be true iff $\{a\} \times (b, c)$ is a ∂P-minimal 2-cell and is contained in $\mathrm{Inv}(P)$.

For P in the example above, $S_{44} = \{(0,1,0,1),(1,2,1,2)\}$, $S_{00} = \mathbf{false}$, and $S_\tau = \emptyset$ for all remaining $\tau \in 5^2$.

For any $\tau \in 5^2$, we can uniformly obtain the finite relation $S\tau$ on ∂P from $\mathrm{Inv}(P)$ and ∂P (and so from P) by means of a FO query.

On the other hand, if we know the finite relations ∂P and all S_τ's, we can uniformly recover $\mathrm{Inv}(P)$ from them, by means of a FO query. As $P = \mathrm{Inv}(P) \cap D$, the same is true for P. So in the binary case we have the result we need.

Actually, the arguments above work not only for $\mathbb{Q}$ but for an arbitrary dense ordered set. However, we will prove the result not only in the dense case, but for an arbitrary linearly ordered set. In that case some extra technical problems with the definition of ∂P arise, because in general for a finitely representable relation the least set over which the relation can be defined does not exist: for example, in $\mathbb{Z}$ the relation $x > 0$ can be represented not only over $\{0\}$ but also over $\{1\}$, because $x > 0$ iff $x \geq 1$. Nevertheless, even in the general case it turns out to be possible to define for any finitely represented relation P a certain canonical finite set of parameters ∂P, over which P can be defined and which can be uniformly obtained from P by means of a FO query.

Now we pass to the general case. We work over an arbitrary (but fixed) linearly ordered set U.

Our aim is to find for finitely representable relations on U, in a sense, a canonical finite representation. We begin with a special case of the so called simple relations.

A relation R on U is said to be *simple* if it can be finitely represented by a disjunction of conjunctions of formulas of the forms $x < c$, $x > c$, or $x = c$, where x is a variable and c is a name of an element of U.

For $k \geqslant 1$, we call sets of the form $I_1 \times \cdots \times I_k$, where each I_j is a singleton or an open interval in U *k-cells* in U. Here an open interval in U is a set defined by a formula of one of the following forms: $x = x$; $a < x < b$; $x < a$; $a < x$. Geometrically, simple k-ary relations over a set B can be described as unions of finitely many k-cells such that all parameters involved in their representations belongs to B. Of course, neither the k-cells nor the set of parameters B are uniquely determined by the simple relation. Simple relations can be characterized as follows.

Lemma 3.5. *For a finite set B, a relation P on U is a simple relation over B iff P is B-invariant.*

Here P is said to be *B-invariant* if $\bar{a} \in P$ iff $\bar{b} \in P$, for any tuples $\bar{a}$ and $\bar{b}$ such that a_i and b_i are positioned the same way with respect to the elements of B, for all i. The proof of the lemma is obvious.

∎

Consider the following binary relation $E_k(\bar{x}, \bar{y})$ on U^k: $x_i < x_j$ iff $y_i < y_j$, for $1 \leqslant i, j \leqslant k$. It is an equivalence relation with finitely many classes. *Fix such a class D.* For $\bar{a} \in D$, the relation $\{(i,j) : a_i = a_j\}$ is an equivalence relation on $\{1, \ldots, k\}$; let $J_1, \ldots, J_s$ be its classes. The relation does not depend on $\bar{a} \in D$.

As usual, for $P \subseteq U^k$, we consider P as a relation of arity k on U. So $P(\bar{a})$ is equivalent to $\bar{a} \in P$. For $P \subseteq D$ and $u, v \in U$, we say that u and v are *P-inseparable* if for any $\bar{a}, \bar{b}$ in D and any $i \in \{1, \ldots, s\}$

$$(\bigwedge_{j \notin J_i} a_j = b_j \wedge \bigwedge_{j \in J_i} (a_j = u \wedge b_j = v)) \to (P(\bar{a}) \equiv P(\bar{b})).$$

Hence, this relation of inseparability is expressible as a restricted query.

For $P \subseteq D$, an element $x \in U$ is said to be a *boundary point* of P if either there is an y such that $y < x$ and for any $y < x$ there is a pair of P-separable elements in $[y, x]$, or there is an y such that $y > x$ and for any $y > x$ there is a pair of P-separable elements in $[x, y]$. Here $[u, v] = \{z : u \leqslant z \leqslant v\}$. Denote by ∂P the set of boundary points of P. It will be observed that any boundary point of P is a constant in every definition of P or is adjacent to such a constant. So the set of the boundary points for each prime relation P is finite and can be expressed as an restricted query.

In Lemmas 3.6–3.9 below, let S be a simple relation defined over a finite set B, and $P = S \cap D$.

Lemma 3.6. *If $x < y$ and $[x, y] \cap B = \emptyset$, there is no pair of P-separable elements in $[x, y]$.*

Proof. Let $u, v \in [x, y]$. Let $\bar{a}, \bar{b} \in D$, and $1 \leqslant i \leqslant s$. Suppose $a_j = b_j$ for $j \notin J_i$, and $a_j = u$, $b_j = v$ for $j \in J_i$. As u and v are positioned in the same way with respect to the elements of B, we have $\bar{a} \in S$ iff $\bar{b} \in S$, and hence $\bar{a} \in P$ iff $\bar{b} \in P$. ∎

Lemma 3.7. *Any boundary point of P is an element of B or is adjacent to an element of B. In particular, ∂P is finite.*

Proof. If z is neither an element of B nor adjacent to an element of B, there are x, y such that $x < z < y$, and $[x, y] \cap B = \emptyset$. Then, by Lemma 3.6, there is no pair of P-separable elements in $[x, y]$. Hence $z \notin \partial P$. ∎

The following lemma is crucial in our consideration.

Lemma 3.8. *If $x < y$ and x, y are P-separable, $[x, y] \cap \partial P \neq \emptyset$.*

Proof. By Lemma 3.6, $[x, y] \cap B \neq \emptyset$. Let $[x, y] \cap B = \{b_1, \dots, b_n\}$, $b_1 < \dots < b_n$. Since x, y are P-separable, the pair x, b_1, or the pair b_n, y, or one of the pairs b_i, b_{i+1} is P-separable. Towards a contradiction, suppose that neither x, y nor the b_i's are boundary points of P. Then b_i and b_{i+1} are P-inseparable, for all i. Indeed, there are $u > b_i$ and $v < b_{i+1}$ such that in $[b_i, u]$ and $[v, b_{i+1}]$ there are no P-separable pairs of elements. If $u \geqslant b_{i+1}$ or $v \leqslant b_i$, we are done. If $b_i < v \leqslant u < b_{i+1}$, the pairs b_i, v and v, b_{i+1} are P-inseparable, and hence the pair b_i, b_{i+1} is P-inseparable, too. In the case $b_i < u < v < b_{i+1}$ the pair u, v is P-inseparable, by Lemma 3.6; since the pairs b_i, u and v, b_{i+1} are P-inseparable, we have the P-inseparability of the pair b_i, b_{i+1}. Analogous arguments show the P-inseparability of each of the pairs x, b_i and b_n, y. A contradiction. ∎

Denote by $\mathrm{Inv}(P)$ the least ∂P-invariant relation containing P. Clearly, $\mathrm{Inv}(P)$ consists of all k-tuples $\bar{b}$ for which there is $\bar{a} \in P$ such that a_i and b_i are positioned the same way with respect to ∂P, for all i. By Lemma 3.5, $\mathrm{Inv}(P)$ is a simple relation over ∂P.

Lemma 3.9. *$P = \mathrm{Inv}(P) \cap D$.*

Proof. We need to prove that there are no $\bar{a} \in P$ and $\bar{b} \in D \setminus P$ such that a_i and b_i are positioned the same way with respect to ∂P, for all i. Suppose there is a counterexample pair $\bar{a}, \bar{b}$. Let $i_1 \in J_1, \dots, i_s \in J_s$; we can assume that $x_{i_1} < \dots < x_{i_s}$ for $\bar{x} \in D$. As $\bar{a} \neq \bar{b}$, there is m such that $a_{i_m} \neq b_{i_m}$, but $a_{i_l} = b_{i_l}$ for $1 \leqslant l < m$. Choose a counterexample pair $\bar{a}, \bar{b}$ with the largest possible m. The points a_{i_m} and b_{i_m} are P-inseparable, by Lemma 3.8, because in the closed interval between them there are no points in ∂P.

Suppose $a_{i_m} < b_{i_m}$. Let $\bar{c} = (c_1, \dots, c_k)$ be the result of replacing in the tuple $\bar{b}$ the elements b_j with a_{i_m}, for all $j \in J_m$. Clearly, $\bar{c} \in D$. Due to the P-inseparability of a_{i_m} and b_{i_m}, we have $\bar{c} \notin P$. The elements a_i and c_i are positioned the same way with respect to ∂P, for all i; so $\bar{a}, \bar{c}$ is a counterexample pair. Since $a_{i_l} = c_{i_l}$ for $1 \leqslant l \leqslant m$, we have a contradiction with the maximality of m.

Now suppose $a_{i_m} > b_{i_m}$. Let $\bar{c} = (c_1, \dots, c_k)$ be the result of replacement in the tuple $\bar{a}$ the elements a_j with b_{i_m}, for all $j \in J_m$. Clearly, $\bar{c} \in D$. Due to the P-inseparability of a_{i_m} and b_{i_m}, we have $\bar{c} \in P$. The elements c_i and b_i are positioned the same way with respect to ∂P, for all i, so $\bar{c}, \bar{b}$ is a counterexample pair. Since $c_{i_l} = b_{i_l}$ for $1 \leqslant l \leqslant m$, we have a contradiction with the maximality of m. ∎

Any k-ary relation P is the disjoint union of all $P \cap D$, where D ranges over the set of E_k-classes. Obviously, if P is finitely representable, every such $P \cap D$ is equal to $S \cap D$, for some simple relation S. So Lemma 3.9 implies

Corollary 3.10. *Any finitely represented k-ary relation P is the disjoint union of all* $\mathrm{Inv}(P \cap D) \cap D$*, where D runs over the set of equivalence classes of E_k.*

∎

Let P be a finitely representable k-ary relation. Denote by ∂P the union of all $\partial(P \cap D)$. As $\mathrm{Inv}(P \cap D)$ is defined over $\partial(P \cap D)$, we have

Corollary 3.11. *Any finitely represented relation P is defined over ∂P.*

∎

Thus, with every finitely represented relation P we have associated a certain canonical finite set of parameters ∂P, over which the relation is defined; the relation P is, in a sense, reduced to a finite family $\{\mathrm{Inv}(P \cap D)\}$ of simple relations over the set ∂P. Moreover, the set ∂P and the family $\{\mathrm{Inv}(P \cap D)\}$ can be found uniformly in P (by means of certain FO queries), and P can be uniformly recovered from the set ∂P and the family $\{\mathrm{Inv}(P \cap D)\}$ (by means of a certain FO query).

Now we are going to find a canonical representation for simple relations. Let S be a simple k-ary relation over a finite set B. For a non-empty B, we call a k-cell $I_1 \times \cdots \times I_k$ *B-minimal* if every I_j can be defined by a formula of one of the following forms:

$$x = b_i; \qquad x < b_1; \qquad b_n < x; \qquad b_i < x < b_{i+1},$$

where $B = \{b_1, \ldots, b_n\}$ and $b_1 < \cdots < b_n$. The only $\emptyset$-minimal k-cell is, by definition, the cell U^k. Obviously, the set of B-minimal k-cells is finite. Clearly, the B-minimal k-cells are pairwise disjoint. Moreover, if a k-cell C' is defined over B and a k-cell C is B-minimal then $C \subseteq C'$ provided $C \cap C' \neq \emptyset$. It follows that S can be decomposed into a disjoint union of B-minimal k-cells, namely, into the disjoint union of all B-minimal k-cells which are contained in S. Note that some of the B-minimal k-cells can be empty.

We show how to encode the simple relation S by a finite family of relations on the finite set B.

Every 1-cell over B is defined by a formula of one of the following forms:

$$(0)\ x = x; \qquad (1)\ x = b; \qquad (2)\ x < b; \qquad (3)\ b < x; \qquad (4)\ b < x < b',$$

where $b, b' \in B$. For $i < 5$, denote by n_i the number of constants from B in the formula (i); so $n_0 = 0$, $n_1 = n_2 = n_3 = 1$, and $n_4 = 2$.

For any $\tau = (\tau_1, \ldots, \tau_k) \in 5^k$, we are going to associate with S and B a relation S_τ on B of arity $n_\tau = n_{\tau_1} + \cdots + n_{\tau_k}$.

Let an n_τ-tuple of variables $\bar{y} = (y_1, \ldots, y_{n_\tau})$ is the concatenation of tuples $\bar{y}_1, \ldots, \bar{y}_k$, where the length of $\bar{y}_i$ is n_{τ_i}. For $i = 1, \ldots, k$, denote by $\phi_i(\bar{x}_i, \bar{y}_i)$ the formula

- $x_i = x_i$ if $n_{\tau_i} = 0$,
- $x_i = v$ if $n_{\tau_i} = 1$ and $\bar{y}_i$ is v,

- $x_i < v$ if $n_{\tau_i} = 2$ and $\bar{y}_i$ is v,
- $x_i > v$ if $n_{\tau_i} = 3$ and $\bar{y}_i$ is v,
- $u < x_i < v$ if $n_{\tau_i} = 4$ and $\bar{y}_i$ is a pair (u, v).

This formula just says that x_i belongs to the 1-cell of type (i) defined by parameters $\bar{y}_i$. Denote by $\phi_\tau(\bar{x}, \bar{y})$ the conjunction of all the ϕ_i's; this formula says that $\bar{x}$ belongs to the k-cell $C_\tau(\bar{y}) = I_1 \times \cdots \times I_k$, where I_i is a k-cell of type τ_i defined by parameters $\bar{y}_i$.

For an n_τ-tuple $\bar{b}$ in U, we define $S_\tau(\bar{b})$ to be true if $\bar{b}$ is in B, and the k-cell $C_\tau(\bar{b})$ is B-minimal and is contained in S. Clearly, the B-minimality of the k-cell means exactly that, for $i \neq 1$, the interval $\phi_i(U, \bar{b}_i)$ has no common points with B.

It is easy to see that the relations S_τ can be uniformly obtained from S and B by means of certain FO queries σ_τ. As S is the union of all k-cells $C_\tau(\bar{b})$ for which $S_\tau(\bar{b})$ holds ($\tau \in 5^k$, $\bar{b} \in B^{n_\tau}$), one can uniformly recover S from B and the family $\{S_\tau\}_{\tau \in 5^k}$ by means of a certain FO query π. Namely, π says that, for one of the τ's, there is an n_τ-tuple $\bar{b}$ in B such that both $S_\tau(\bar{b})$ and $\phi_\tau(\bar{x}, \bar{b})$ hold.

Later we will need the following observation concerning the definition of π.

Observation. *Suppose $A = \{a_1, \ldots, a_m\} \subseteq U$ with $a_1 < \cdots < a_m$, and R_τ are arbirary finite relations of arity n_τ on A. Then it is easy to write down a quantifier-free formula $\rho(\bar{x}, z_1, \ldots, z_m)$ in the pure order language depending only on the isomorphism type of the finite structure*

$$\mathfrak{A} = (A, R_\tau, <, a_1, \ldots, a_m)_{\tau \in 5^k},$$

which says that, for one of the τ's, there is an n_τ-tuple $\bar{z}$ in the set

$$\{z_1, \ldots, z_m\}$$

such that both $R_\tau(\bar{z})$ and $\phi_\tau(\bar{x}, \bar{z})$ hold. So we can assert not only that the relation $\pi(A, \{R_\tau\})$ is finitely representable over A but, moreover, that there is a certain 'standard' finite representation for it over A which depends only on the isomorphism type of $\mathfrak{A}$.

∎

Based on the analysis above and taking into account Corollary 3.10, it is an easy exercise to prove

Lemma 3.12. *Consider the following two database schemes η and θ. The scheme η consists of one k-ary symbol P; the scheme θ consists of a unary symbol B and n_τ-ary symbols $S_{D\tau}$, for $\tau \in 5^k$ and E_k-classes D.*

(1) *There is a unary FO η-query δ which, for any finitely represented k-ary input P, yields a finite set ∂P as the answer*
(2) *For any $\tau \in 5^k$ and any E_k-class D, there is a n_τ-ary FO η-query $\sigma_{D\tau}$ which, for any finitely represented k-ary input P, yields the finite relation $\mathrm{Inv}(P \cap D)_\tau$ on ∂P as the answer*

(3) *There is a k-ary FO θ-query π which for any finite input*

$$B, \{S_{D\tau} : \tau \in 5^k, D \text{ is a } E_k\text{-class}\},$$

yields as an answer a relation finitely representable over B.

(4) *The family of queries δ, $\{\sigma_{D\tau}\}$ is an inverse for the query π in the following sense: for any finitely represented k-ary input P, we have $P = \pi(\delta(P), \{\sigma_{D\tau}(P)\})$*

∎

We summarize the consideration above in the following main results.

Theorem 3.13. *For any finite database scheme $\eta = \{P_1, \dots, P_n\}$ there are:*

- *a database scheme $\theta = \{B, S_1, \dots, S_m\}$, where B is unary*
- *η-queries δ of arity 1 and σ_i of arity of S_i, for $1 \leqslant i \leqslant m$*
- *locally generic θ-queries π_j of arity of P_j, for $1 \leqslant j \leqslant n$*

such that

(a) *for any finitely representable η-input p, the family of η-queries $\bar{\sigma} = \{\delta, \sigma_i\}$ yields a finite θ-state as the answer*

(b) *for any finite θ-state s, the family of θ-queries $\bar{\pi} = \{\pi_i\}$ yields as the answer an η-state finitely representable over B*

(c) *for any η-query, which is locally generic over finitely representable states, the result of replacing of $P_1, \dots, P_n$ in it with $\pi_1, \dots, \pi_n$ is locally generic over finite θ-states*

(d) *$\bar{\pi}$ is an inverse of $\bar{\sigma}$ in the following sense: $\bar{\pi}(\bar{\sigma}(p)) = p$, for any finitely representable η-state p*

∎

Note that (c) here immediately follows from the observation above concerning the definition of π.

Theorem 3.14. *For any expanded ordered universe U, if*

(1) *for any finite database scheme ρ, any locally generic over finite states extended ρ-query is equivalent over finite states over U to a restricted ρ-query*

then

(2) *for any finite database scheme η, any locally generic over finite representable states extended η-query is equivalent over finite representable states over U to a restricted η-query*

Proof. Suppose (1) is true. Fix a finite database scheme $\eta = \{P_1, \dots, P_n\}$. Let Φ be a locally generic over finitely representable states extended η-query.

We use Theorem 3.13. First we replace in Φ the relation names $P_1, \dots, P_n$ with the formulas $\pi_1, \dots \pi_n$; we obtain a locally generic over finite states extended θ-query Ψ. By (1), the query Ψ is equivalent over finite states to a restricted θ-query Ψ_1. Now we replace in Ψ_1 the relation names $B, S_1, \dots, S_m$

with the formulas $\delta, \sigma_1, \ldots, \sigma_m$. Then we obtain a restricted η-query Φ_1 equivalent over finite representable states to Φ. So we have proven (2). ∎

3.4. **Collapse of extended locally generic queries.** In this section, we pursue collapse results over finite states. However, all the results can be transferred to finitely representable states by directly applying Theorem 3.14.

For convenience, consider database schemes that contain not only relation symbols, but also finitely many constant symbols. A database state over a universe U for such a scheme is a mapping that assigns to any relation symbol in the scheme a relation on U of the corresponding arity, and to any constant symbol in the scheme an element in U. In this case the active domain of a database state is defined to be the union of the active domain of the relational part of the state and the set of values of all constants of the scheme. For a relational database scheme SC, denote by SC_k the scheme $SC \cup \{c_1, \ldots, c_k\}$, where the c_i are new constant symbols.

Clearly, two k-ary SC-queries

$$\phi(x_1, \ldots, x_k) \quad \text{and} \quad \psi(x_1, \ldots, x_k)$$

are equivalent over finite states over a universe U if the Boolean SC_k-queries

$$\phi(c_1, \ldots, c_k) \quad \text{and} \quad \psi(c_1, \ldots, c_k)$$

are equivalent over finite states over U.

The notions of genericity and local genericity for SC_k-queries are defined exactly the same way as for SC-queries. Clearly, a k-ary SC-query $\phi(x_1, \ldots, x_k)$ is generic (locally generic) over U iff the Boolean SC_k-query $\phi(c_1, \ldots, c_k)$ is generic (locally generic) over U.

Our ultimate goal is to prove that, under certain conditions on the universe U, any locally generic extended query is equivalent over finite states over U to a restricted query. Hence, it suffices to prove such a result for Boolean queries (for database schemes with constant symbols).

For an arbitrary signature L, an L-theory is defined to be a set of first order L-sentences (that is, formulas of signature L without free variables). For a class K of structures of an arbitrary signature L (in symbols, L-structures), the first order L-theory of K (in symbols, $\mathrm{Th}(K)$) is defined to be the set of all first order L-sentences which hold in every structure in K. Two L-structures M and N are called *elementarily equivalent* (in symbols, $M \equiv N$), if ϕ holds in M iff ϕ holds in N, for any L-sentence ϕ. An L-theory T is said to be complete if all its models are elementarily equivalent.

Let ρ be a database scheme $\{R_1, \ldots, R_n, c_1 \ldots, c_k\}$. We denote $L \cup \rho$ by $L(\rho)$. A ρ-state s over an L-structure W is said to be *pseudo-finite in* W if (W, s) is a model of the first order $L(\rho)$-theory $F(W, \rho)$ of all (W, r), where r is a finite ρ-state over W.

For a first order $L(\rho)$-sentence ψ and $m < \omega$, there is a first order L-sentence ψ_m such that, for any L-structure V, the sentence ψ_m holds in V iff ψ holds for all ρ-states over V, whose active domain has cardinality at most m. Thus, $\psi \in F(W,\rho)$ iff $\{\psi_m : m < \omega\} \subseteq \mathrm{Th}(W)$. It follows that $W \equiv V$ implies $F(V,\rho) = F(W,\rho)$. For a complete L-theory T, the first order $L(\rho)$-theory $F(T,\rho)$ is well-defined to be $F(W,\rho)$, where W is an arbitrary model of T.

Let ρ' be a disjoint copy $\{R'_1,\dots,R'_n,c'_1\dots,c'_k\}$ of ρ. For an $L(\rho)$-sentence θ denote by $\theta(\rho')$ its $L(\rho')$-copy, that is, the result of replacement of every occurrence of R_i and c_j in θ with R'_i and c'_j, respectively. Let $\bar\rho = \rho \cup \rho'$.

We say that a Boolean extended ρ-query ϕ is *generic for pseudo-finite states in* V if the following holds: for any $\bar\rho$-state (r,r'), if (r,r') is pseudo-finite in V and r can be transformed to r' by an L_0-automorphism of V, then $\phi(\rho)$ holds in (V,r) iff $\phi(\rho')$ holds in (V,r').

As we will use the standard technique of so-called special models, we summarize its basic definitions and facts (see [CK90] for detail).

For a structure M of an arbitrary signature L and a subset A of M, denote by $L(A)$ the signature obtained by adjoining to L names for the elements of A. We do not normally distinguish between elements of A and their names.

We say that M is an *elementary substructure* of N (in symbols, $M \preceq N$ or $N \succeq M$), if M is a substructure of N, and ϕ holds in M iff ϕ holds in N, for any $L(M)$-sentence ϕ.

A set p of first order $L(A)$-formulas with one free variable x is said to be a *type* over A in M if every finite subset $\{\phi_1(x),\dots,\phi_k(x)\}$ of p is realized in M (that is, $(\exists x)(\phi_1(x) \& \dots \& \phi_k(x))$ holds in M), and, for every $L(A)$-formula $\phi(x)$, either $\phi \in p$ or $\neg\phi \in p$. We say that a subset q of p *isolates* p if p is the only type over A in M containing q.

Let A be a subset of M. For any $N \succeq M$ and $a \in N$, the set of all $L(A)$-formulas $\phi(x)$ such that $\phi(a)$ holds in N forms a type over A in M; denote it by $\mathrm{tp}(a/A)$. For any type p over A in M, there are $N \succeq M$ and $a \in N$ such that $p = \mathrm{tp}(a/A)$. We denote $\mathrm{tp}(A)$ the set of all $L(A)$-sentences which hold in M.

For a cardinal λ, a structure M is said to be λ-*saturated* if any type p over any its subset A of power $< \lambda$ is realized in M; that is, $p = \mathrm{tp}(a/A)$, for some $a \in M$. For any infinite $\lambda \geqslant |L|$, every two elementarily equivalent λ-saturated structures of power λ are isomorphic.

A structure M of power λ is called *special* if M is the union of a family $\{M_\mu : \mu \text{ is a cardinal } < \lambda\}$, where $M_\mu \preceq M_\nu \preceq M$ for $\mu < \nu < \lambda$, and each M_μ is μ^+-saturated. Here μ^+, as usual, denotes the least cardinal greater than μ. Every two elementarily equivalent special structures of the same power are isomorphic. For any infinite L-structure M and any cardinal $\lambda > |L|, |M|$ with $\lambda^* = \lambda$, there exists a special $N \succeq M$ of power λ. Here $\aleph^*_\alpha$ is defined to be $\sum_{\beta<\alpha} 2^{\aleph_\beta}$. It is easy to construct cardinals λ with $\lambda^* = \lambda$ of arbitrarily large cofinality.

Theorem 3.15. *For any countable universe U and any Boolean extended ρ-query ϕ the following conditions are equivalent:*

(1) *there is a restricted ρ-query ψ which is equivalent to ϕ over finite database states over U*
(2) *ϕ is generic for pseudo-finite states over V, for all $V \equiv U$*
(3) *for every uncountable power κ with $\kappa = \kappa^*$, the query ϕ is generic over pseudo-finite states over the special model $V \equiv U$ with $|V| = \kappa$*

∎

Let I be a subset of a universe V. We say that a Boolean extended ρ-query ϕ is *locally generic over pseudo-finite states over I in V* if the following holds: if an $\bar{\rho}$-state (r, r') over I is pseudo-finite in V and r can be transformed into r' by a partial L_0-isomorphism in V then $\phi(\rho)$ holds in (V, r) iff $\phi(\rho')$ holds in (V, r').

A linearly ordered subset I of a structure M is said to be an *indiscernible sequence* in M if $\theta(\bar{a})$ holds in M iff $\theta(\bar{b})$ holds in M, for every first order L-formula $\theta(x_1, \ldots, x_n)$ and any two n-tuples $\bar{a}$ and $\bar{b}$ in I with $a_1 < \cdots < a_n$ and $b_1 < \cdots < b_n$.

Theorem 3.16. *Let an extended Boolean ρ-query ϕ be locally generic for finite states over U. Suppose, for some uncountable κ with $\kappa = \kappa^*$, there is a special model $V \equiv U$ of power κ such that, for any infinite indiscernible sequence I in V, the query ϕ is locally generic over pseudo-finite states over I in V. Then ϕ is equivalent over finite states over U to a restricted ρ-query.*

∎

As a side remark, note that this technique implies a result from [BDLW96] and [OV95] about the so-called *active semantics.* An FO formula is said to be *active* if it is obtained from some formula by relativization of every of its quantifiers with respect to the formula $AD(x)$.

Theorem 3.17. *For finite states, any locally generic active extended query is equivalent to a restricted query.*

Proof. By Theorem 3.16, it suffices to show that any active Boolean query is locally generic over *arbitrary* states over any indiscernible sequence in any V. We may assume that the signature L is relational. For a ρ-state r over V, denote by V_r the substructure of (V, r) with domain $ad(r)$. Clearly, an active ρ-query holds in V_r iff it holds in (V, r). Since, for any ρ-states r and r' over an indiscernible sequence I in V, any partial L_0-isomorphism transforming r into r' induces an $L(\rho)$-isomorphism between V_r and $V_{r'}$, the result follows. ∎

Now our aim is to give a general condition on a complete theory which ensures collapse of locally order-generic queries to pure order ones, over finite states over models of T.

Let M be an L-structure, and A, B be subsets of M. A bijection $h : A \to B$ is said to be a *partial L-isomorphism* (an *elementary map*) in M if $\phi(\bar{a})$ holds in M iff $\phi(h(\bar{a}))$ holds in M, for any atomic L-formula (for any L-formula) $\phi(\bar{x})$ and any tuple $\bar{a}$ in A. Clearly, every elementary map in M is a partial isomorphism in M; if M admits quantifier elimination, the converse is also true.

We recall the following well-known result (see [Sac72], [RW81]):

Fact 3.18. *Let T be a complete L-theory. Then the following are equivalent:*

(1) *T admits quantifier elimination;*

(2) *there is an infinite cardinal λ such that, for every finitely generated L-structure A, for every models M, N of T with $A \subseteq M, N$, for every $a \in N \setminus A$, if M is λ-saturated, the quantifier-free type of a is realized in M;*

(3) *there is an infinite cardinal λ such that every λ-saturated model M of T is finitely homogeneous in the following sense: for any partial isomorphism $h : A \to B$ in M with finite A and B, and any $a \in M$ there is $b \in M$ such that $h \cup \{(a, b)\}$ is a partial isomorphism in M.*

∎

Let T be an L-theory, and T' be an L'-theory with $L \subseteq L'$ and $T \subseteq T'$. The theory T' is said to be a *definitional expansion* of T if every L'-formula is equivalent in T' to an L-formula. Every L-theory T has a *standard* definitional expansion admitting quantifier elimination: for every L-formula $\phi(\bar{x})$ with n free variables add a new n-ary relation symbol P_ϕ to L and a new axiom $\forall \bar{x}\, (P_\phi(\bar{x}) \leftrightarrow \phi(\bar{x}))$ to T.

We say that a complete L-theory T has the *Pseudo-finite Homogeneity Property* if there exist its definitional expansion T' and an infinite cardinal λ such that, whenever A and B are pseudo-finite sets in a model M' of T', and $h : A \to B$ is a partial isomorphism in M' with λ-saturated (M', A, B, h), for any $a \in M'$ there is $b \in M'$ such that $h \cup \{(a, b)\}$ is a partial isomorphism in M'.

Note that, by the Fact above, the theory T' here automatically admits quantifier elimination, because any finite set in M' is pseudo-finite, and if M' is λ-saturated then (M', A, B, h) is λ-saturated, too, for finite A and B.

Therefore the definition of the Pseudo-finite Homogeneity Property admits the following equivalent formulation: a complete L-theory T has the Pseudo-finite Homogeneity Property iff there exists an infinite cardinal λ such that, whenever A and B are pseudo-finite sets in a model M of T, and $h : A \to B$ is an elementary map in M with λ-saturated (M, A, B, h), for any $a \in M$ there is $b \in M$ such that $h \cup \{(a, b)\}$ is an elementary map in M.

The Pseudo-finite Homogeneity Property makes sense not only for theories of ordered universes, and there are obvious examples of theories with this property. For example, in infinite structures of empty signature the pseudo-finite sets are exactly coinfinite sets; hence the Pseudo-finite Homogeneity Property for the theory of these structures easily follows. It turned out that for ordered structures the property is especially interesting because it gives a sufficient condition for collapse results. Later we will give a series of examples of ordered universes with this property.

Theorem 3.19. *Suppose the first order theory of a universe U has the Pseudo-finite Homogeneity Property. Let an extended query ϕ be locally generic over finite states over U. Then ϕ is equivalent over finite states over U to a restricted query.*

Proof. It suffices to prove that ϕ satisfies the condition of Theorem 3.16. Suppose λ witnesses that T has the Pseudo-finite Homogeneity Property. Let $\kappa = \kappa^* > |L| + \aleph_0$, and $\mathrm{cf}(\kappa) > \lambda$. Let $V \equiv U$ be a special model of power κ. Let I be an infinite L-indiscernible sequence in V. Suppose $\bar{\rho}$-state (r, r') over I is pseudo-finite in V and r can be transformed to r' by a partial L_0-isomorphism g in V, whose domain is A, the active domain of r, and whose range is A', the active domain of r'. We need to show that $\phi(\rho)$ holds in (V, r) iff $\phi(\rho')$ holds in (V, r'). We may assume that (V, A, A', g) is λ-saturated. (Indeed, consider a special model $(V_0, r_0, r'_0, g_0, I_0)$ of power κ elementarily equivalent to (V, r, r', g, I). It suffices to prove the claim for $(V_0, r_0, r'_0, g_0, I_0)$; but it is λ-saturated as $\mathrm{cf}\kappa > \lambda$.) Using a Fraïssé-Ehrenfeucht game, we will show that g is an $L(\rho)$-elementary map from (V, r) to (V, r'). Due to the L-indiscernibility of I, the map g is an L-elementary map, and, in particular, a partial $L(\rho)$-isomorphism. Therefore, due to the Pseudo-finite Homogeneity Property, to complete the proof of the theorem, using Theorem 3.16, it suffices to prove the following lemmas:

Lemma 3.20. *The active domain of any pseudo-finite database state is a pseudo-finite set.*

Lemma 3.21. *Let A be a pseudo-finite set in V, and $a \in V$. Then $A \cup \{a\}$ is a pseudo-finite set.*

Lemma 3.22. *If $h : C \to D$ is a partial isomorphism in V, and $M = (V, C, D, h)$ is λ-saturated, then*

$$M' = (V, C \cup \{c\}, D \cup \{d\}, h \cup \{(c, d)\})$$

is λ-saturated, for any $c, d \in V$.

Proof of Lemma 3.20. Consider the database scheme $\tau = \{P\}$, where P is a unary relation name. For any $L(\tau)$-sentence γ, there is an $L(\rho)$-sentence γ^* such that $(V, s) \models \gamma^*$ iff $(V, ad(s)) \models \gamma$, for any ρ-state s. Suppose a state s is pseudo-finite in V, and $\gamma \in F(V, \tau)$. Since the active domain of any finite state is finite, we have $(V, r) \models \gamma^*$ for all finite ρ-states r. So $(V, s) \models \gamma^*$, and hence $(V, ad(s)) \models \gamma$. ∎

Proof of Lemma 3.21. Consider the database scheme $\tau = \{P\}$, where P is a unary relation name. Let $\theta \in F(V, \tau)$. Let $\theta^*(x)$ be the result of replacement of every occurrence of $P(y)$ in θ with $P(y) \vee y = x$, where x is a new variable. Then $\forall x \theta^*(x)$ belongs to $F(V, \tau)$ and so holds in (V, A). Hence θ holds in $(V, A \cup \{a\})$. Thus, $A \cup \{a\}$ is pseudo-finite. ∎

Proof of Lemma 3.22. M' is definable in the λ-saturated structure M with parameters c, d. ∎

The collapse result is proved. ∎

Now we introduce a certain property of complete theories which is strictly stronger than the Pseudo-finite Homogeneity Property, and so ensures the collapse result, too.

We say that a complete theory T has the *Isolation Property*, if there is a cardinal λ such that, for any pseudo-finite set A and any element a in a model of T, there is $A_0 \subseteq A$ with $|A_0| < \lambda$ such that $\mathrm{tp}(a/A_0)$ isolates $\mathrm{tp}(a/A)$.

Theorem 3.23. *The Isolation Property implies the Pseudo-finite Homogeneity Property.*

Proof. We show that λ witnessing that T has the Isolation Property witnesses that T has the Pseudo-finite Homogeneity Property. Let V be a λ-saturated model of T, and A, B pseudo-finite sets in V, and $a \in V$. Let $h : A \to B$ be an L-elementary map in V. We show that there is $b \in B$ such that $h \cup \{(a, b)\}$ is an L-elementary map in V. Choose $A_0 \subseteq A$ with $|A_0| < \lambda$ such that $p_0 = \mathrm{tp}(a/A_0)$ isolates $p = \mathrm{tp}(a/A)$. Since the map h is elementary, $h(p)$ is a type over B, and $h(p_0)$ isolates $h(p)$. (For a set $q(x)$ of formulas over A we denote by $h(q)$ the set $\{\theta(x, h(\bar{c})) : \theta(x, \bar{c}) \in q\}$.) As V is λ-saturated, there is $b \in V$ realizing $h(p_0)$ and hence $h(p)$. So $h \cup \{(a, b)\}$ is an L-elementary map. ∎

It can be shown that any theory with the Isolation Property is unstable (the latter means that there are $n \geq 1$, a formula $\phi(\bar{x}, \bar{y})$, where $\bar{x}, \bar{y}$ are n-tuples of free variables, and an infinite sequence $\bar{a}_0, \bar{a}_1, \ldots$ of n-tuples in a model of the theory such that $\phi(\bar{a}_i, \bar{a}_j)$ holds iff $i < j$.) On the other hand, there obvious examples of stable theories with Pseudo-finite Homogeneity Property (for example, the theory of infinite structures of empty signature). Later we will give examples of ordered universes which have the Pseudo-finite Homogeneity Property but do not have the Isolation Property (they are, of course, unstable). Theorem 3.26 below gives a broad class of theories with the Isolation Property and provides a lot of examples of collapse results.

A complete L-theory T is said to be *o-minimal* if in every model of T any definable set is a finite union of singletons and open intervals. The following characterization of o-minimal theories is not hard to prove.

Theorem 3.24. *A complete L-theory T is o-minimal iff there exists T', a definitional expansion of T in a language L', such that any L'-formula $\theta(x, \bar{y})$ is T'-equivalent to a disjunction of formulas of the form $\psi(\bar{y}) \wedge \rho(x, \bar{y})$, where $\rho(x, \bar{y})$ has one of the following forms, for some L'-terms t and t' in the variables $\bar{y}$:*

$$x = x, \quad x = t, \quad x < t, \quad t < x, \quad t < x < t'.$$

∎

We call a complete L-theory T *quasi-o-minimal* iff there exists T', a definitional expansion of T in a language L', such that any L'-formula $\theta(x, \bar{y})$ is T'-equivalent to a disjunction of formulas of the form $\phi(x) \wedge \psi(\bar{y}) \wedge \rho(x, \bar{y})$, where $\rho(x, \bar{y})$ has one of the following forms, for some L'-terms t and t' in the variables $\bar{y}$:

$$x = x, \quad x = t, \quad x < t, \quad t < x, \quad t < x < t'.$$

By Theorem 3.24, every o-minimal theory is quasi-o-minimal. Clearly, in every model of a quasi-o-minimal theory any definable set is a finite union of singletons and sets of the form $I \cap D$, where I is an open interval and D is a set definable without parameters.

There exist quasi-o-minimal theories which are not o-minimal. The simplest example is the theory T of dense ordered sets without endpoints with a distinguished subset which is dense and codense in the universe. It can be easily shown that T is the theory of the structure $(\mathbb{R}, <, \mathbb{Q})$. The theory is not o-minimal because the distinguished subset is not a finite union of singletons and open intervals. Standard arguments show that T admits quantifier elimination; obviously, we can take T as T' from the definition of quasi-o-minimality.

Another example of a quasi-o-minimal theory is the theory T of $(\mathbb{Z}, <, +)$. Indeed, by Presburger's Theorem, the definitional expansion of the model by the constants 0, 1 and the unary predicates 'n divides x', for all positive integers n, admits quantifier elimination. For a positive integer n, define the function $f_n(x)$ by the condition $0 \leqslant x - nf_n(x) < n$. Since '$n$ divides x' iff $nf_n(x) = x$, and $nx = t$ iff $x = f_n(t)$, and $nx < t$ iff $x < f_n(t)$, and $nx > t$ iff $x > f_n(t)$, the theory T' of the definitional expansion of $(\mathbb{Z}, <, +)$ by the constants 0, 1 and all the functions $f_n(x)$ satisfies the condition of the definition of quasi-o-minimality. However, the theory of $(\mathbb{Z}, <, +)$ is not o-minimal because the definable subsets $n\mathbb{Z}$ are not finite unions of singletons and open intervals.

Similarly, the theory of $(\mathbb{N}, <, +)$ is quasi-o-minimal but not o-minimal.

New examples of quasi-o-minimal structures can be constructed using the *ordered union* operation; the operations of this type are popular in database applications. Let U_i be an L_i-structure, where L_i contains $<$, and U_i is linarly ordered by $<$, for $i = 1, 2$. We assume that the universes of U_1 and U_2 are disjoint. Let $L = L_1 \cup L_2 \cup \{P_1, P_2\}$, where P_1 and P_2 are new unary relation names. The ordered union of U_1 and U_2 is defined to be the L-structure U, whose universe is the union of the universes of U_1 and U_2, and

- P_i^U is defined to be the universe of U_i,
- $<^U$ is defined to be $<^{U_1} \cup <^{U_2} \cup (U_1 \times U_2)$,
- $Q^U = Q^{U_i}$ for any $Q \in L_i \setminus \{<\}$.

A routine induction on the complexity of a formula shows that any L-formula is equivalent in U to a positive Boolean combination of L-formulas in each of which all variables, free or bounded, are restricted to P_1 or are restricted to P_2. It follows, in particular, that the models of the theory of U are exactly the ordered unions of U_1' and U_2', where $U_1 \equiv U_1'$ and $U_2 \equiv U_2'$. So the ordered union of two complete theories of ordered structures can be well-defined. Another obvious consequence is

Theorem 3.25. *The ordered union of any two quasi-o-minimal theories is a quasi-o-minimal theory.*

∎

Note that the ordered union of two o-minimal theories is an o-minimal theory iff any model of the first of them has a greatest element or any model of the second of them has a least element. So the construction gives a lot of quasi-o-minimal theories which are not o-minimal.

Theorem 3.26. *Any quasi-o-minimal theory has the Isolation Property.*

∎

Now we give a series of examples of ordered structures which show that the Isolation Property is strictly weaker than quasi-o-minimality and strictly stronger than the Pseudo-finite Homogeneity Property.

Let $L = \{<, E\}$ and T_{dt} be the theory of all the structures of the form $(A, <, E)$, where $<$ is a dense ordering without endpoints, and E is an equivalence relation with two class both of which are dense in A. The structure $(\mathbb{R}, <, E)$, where E is the equivalence relation whose classes are $\mathbb{Q}$ and $\mathbb{R} \setminus \mathbb{Q}$, is a model of T_{dt}; so T_{dt} is consistent. Standard back-and-forth arguments show that T_{dt} is countably categorical and hence complete. Also, standard arguments show that T_{dt} admits quantifier elimination.

Theorem 3.27. *T_{dt} is not quasi-o-minimal, but has the Isolation Property.*

∎

Consider the structures of the form $(A, <, E)$, where $(A, <)$ is a dense linearly ordered set without endpoints, and E is an equivalence relation on A with infinitely many classes all of which are dense. An example of such a structure is $(\mathbb{R}, <, E)$, where $E(x, y)$ means $x - y \in \mathbb{Q}$. Standard back-and-forth arguments show that the theory T_{de} of such structures is countably categorical and hence complete. Also, standard arguments show that T_{de} admits quantifier elimination.

Theorem 3.28. *T_{de} does not have the Isolation Property, but has the Pseudo-finite Homogeneity Property in the following strong sense: for every model M of T_{de}, whenever A and B are pseudo-finite sets in M, and $h : A \to B$ is a partial isomorphism in M, for any $a \in M$ there is $b \in M$ such that $h \cup \{(a, b)\}$ is a partial isomorphism in M.*

∎

Let F be an ordered division ring, T_0 be the first order theory of ordered vector spaces over F with a distinguished subspace, and T_{ds} be the first order theory of ordered nonzero vector spaces over F with a distinguished proper dense subspace. Here an ordered vector space over F is defined to be a vector space V over F whose additive group is linearly ordered so that αv is positive, for any positive $\alpha \in F$ and any positive $v \in V$. We consider T_0 and T_{ds} in the signature $\{+, <, f_\alpha, P\}_{\alpha \in F}$, where f_α is a name for the unary operation of multiplication by the scalar α, and P is a name for the distinguished subspace. The theory T_0 is obviously consistent. We will show the consistency of T_{ds} in the proof of Theorem 3.29 below.

A first order theory T is said to be *model complete* iff for all models A and B of T, if $A \subset B$ then $A \preceq B$. Clearly, if a theory admits quantifier elimination, it is model complete. A theory T^* is said to be a *model completion* of it subtheory T if, firstly, any model of T can be embedded into a model of T^*, and, secondly, T^* is complete over any model of T, that is, for any model A of T and any models B, C of T^* with $A \subseteq B, C$, the structures $(B, a)_{a \in A}$ and $(C, a)_{a \in A}$ are elementarily equivalent.

Theorem 3.29. *T_{ds} admits quantifier elimination, is complete, and is a model completion of T_0.*

∎

Theorem 3.30. *The theory T_{ds} has the Pseudo-finite Homogeneity Property (with $\lambda = |F|^+$), but doesn't have the Isolation Property.*

∎

The general setting we considered really gives some concrete examples of collapse results. For instanse, the collapse result holds for any structure of the form $(\mathbb{R}, +, <, F, f_\alpha)_{\alpha \in F}$, where F is a subfield of $\mathbb{R}$.

Note that for the structure $(\mathbb{R}, +, \times, <, \mathbb{Q})$ the collapse result fails.

Firstly, the locally generic query "*the number of elements of P is even*" over $M = (\mathbb{R}, +, \times, <, \mathbb{Q})$ is expressible in the first order extended language. For example, the cardinality of a subset P of $\mathbb{R}$ is even iff (M, P) satisfies the first order sentence which says:

> *If $P \neq \emptyset$, there is a real number α such that the integral parts $[x]$ of elements $x \in \alpha P$ are pairwise distinct, and for some even positive integer n and some integer m, the remainders when m is divided by $n! + 1,\ 2(n!) + 1,\ \ldots,\ n(n!) + 1$ are pairwise dictinct and form the set $\{[x] : x \in \alpha P\}$.*

(The latter sentence is first order indeed: as $\mathbb{Z}$ is first order definable in the field of the rational numbers without parameters (see [Rob49]), it is definable in M, too.) If the sentence holds, the cardinality of P is obviously even. Suppose P is not empty, the cardinality of P is n, and n is even. Choose a nonzero real α so that $|x - y| \geqslant 1$ for any different $x, y \in \alpha P$. Then $[x]$ are pairwise distinct, for $x \in \alpha P$. Let $\{[x] : x \in \alpha P\} = \{r_1, \dots, r_n\}$. As $n! + 1$, $2(n!) + 1$, $\dots$, $n(n!) + 1$ are pairwise coprime, by the Chinese remainder theorem, there is an integer m such that for $1 \leqslant i \leqslant n$, the remainder when m is divided by $i(n!) + 1$ is r_i, and we are done.

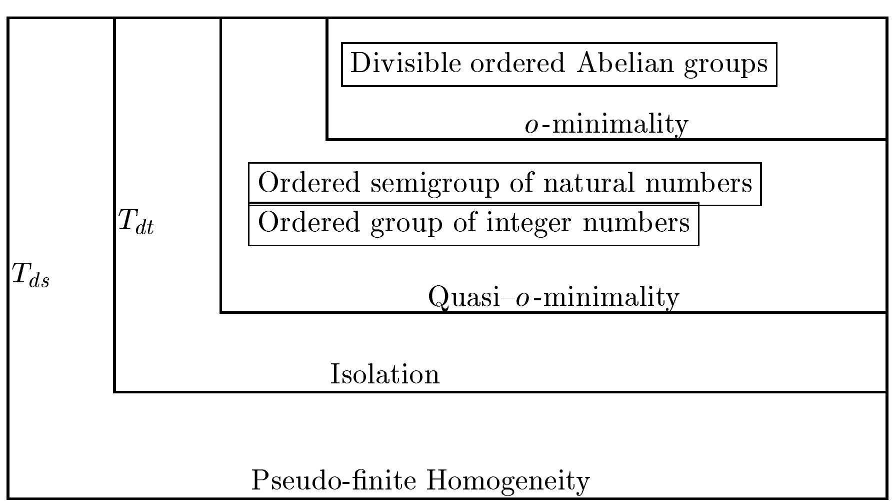
Secondly, on the other hand, the sentence cannot be expressed as a restricted query, otherwise, by compactness, we could construct two elementarily nonequivalent dense ordering without endpoints with distinguished infinite pseudo-finite subsets; this is impossible as shown in the proof of Theorem 3.4.

A modification of the argument above shows that the locally generic query "*the cardinality of P is even*" over the ordered field of rationals is expressible in the first order extended language, even though we know that it cannot be expressed as a restricted query — the latter can be shown by the same arguments as above. The cardinality of a set of rational numbers P is even iff $(\mathbb{Q}, +, \times, <, P)$ satisfies the first order sentence which says:

> *If $P \neq \emptyset$, there is a positive integer k such that $kP \subseteq \mathbb{Z}$ and, for some even positive integer n and some integer m, the remainders when m is divided by $n! + 1$, $2(n!) + 1$, $\dots$, $n(n!) + 1$ are pairwise dictinct and form the set kP.*

Indeed, if the sentence holds, the cardinality of P is obviously even. Suppose P is not empty, the cardinality of P is n, and n is even. Let k be the product of the denominators of members of P; then $kP \subseteq \mathbb{Z}$. Let $kP = \{r_1, \dots, r_n\}$. There is an integer m such that for $1 \leqslant i \leqslant n$ the remainder when m is divided by $i(n!) + 1$ is r_i, and we are done.

The following picture presents our collapse results.



4. Stratified Datalog

4.1. **Definitions.** Throughout this last part of the paper, we deal with the domain $\mathbb{Z}$ of integer numbers together with the relations $=$ of equality, $<$ of the integer linear order, and $<_g$ of the integer gap order, for all natural numbers g. Here $x <_g y$ means that $x + g < y$. The gap orders are first-order expressible using usual order but the reason to include them is that the resulting first-order language admits quantifier elimination. As $x < y$ is equivalent to $x <_0 y$, we are able not to use usual order. Atomic formulas that use equality, order or gap order only are called pure domain formulas, or *atoms.* The truth values of atoms is defined in the usual way.

$Datalog^{\neg,<_z}$-programs also use extra (not pure domain) relation names, each of a fixed arity. The extra names are of two types. The names of the first type are *extensional,* or *input,* names. They are thought of as the input database.

The names of the second type are *intensional,* for their meaning is going to be computed by our program. Further, some of the intensional names are called *output* names, and the remaining intensional names are called *internal.* The idea is, although we are interested in computing output relations only, our computation itself may require generating intermediate results that are temporarily stored in internal names and discarded afterwards.

Then, we want to consider the stratified negation. That is to say, intensional names may be used under negation, but not sooner than their calculation terminates. Formally, intensional names are ranked by consecutive positive integer numbers with the smallest rank 1. Several intensional names can have the same rank. An intensional name may be used under negation only in defining an intensional name of a higher rank.

The syntax of $Datalog^{\neg,<_z}$ is traditional. A $Datalog^{\neg,<_z}$-program, which we sometimes also refer to as a *stratified program,* is a finite set of rules. Each rule has a head and a body. The body can be either empty or be a sequence of formulas. The head of a rule is an atomic formula of the form $P(x_1, x_2, \ldots, x_n)$ where $x_1, x_2, \ldots, x_n$ is a list of pairwise different variables and P is an intensional name of arity n. The rank of the rule is defined to be the rank of P. Each formula in a body must be of one of the following form:

- an atom or its negation
- an atomic formula of the form $P(x_1, x_2, \ldots, x_k)$, where $x_1, x_2, \ldots, x_k$ is a list of variables and P is an extensional name of arity k
- an atomic formula of the form $P(x_1, x_2, \ldots, x_k)$, where $x_1, x_2, \ldots, x_k$ is a list of variables and P is a k-ary intensional name *whose rank is less than or equal to the rank of the rule*
- a formula of the form $\neg P(x_1, x_2, \ldots, x_k)$ where $x_1, x_2, \ldots, x_k$ is a list of variables and P is a k-ary intensional name *whose rank is less than the rank of the rule*

So a rule has the form

$$P(x_1, x_2, \ldots, x_n) \longleftarrow \varphi_1, \varphi_2, \ldots, \varphi_m,$$

where $\varphi_1, \varphi_2, \ldots, \varphi_m$ are formulas used in its body. Note that we do not require the variables occurring at the right side to occur at the left side; in case they do not, the interpretation is existential — see a formal definition below.

A *state* for a $Datalog^{\neg,<_z}$-program R is an assignment of a set of integer number tuples of proper arity to every extensional name in R. The set of tuples assigned by a state to an extensional name is the input of the state for the extensional name. We assume that all the sets are represented finitely by quantifier-free pure domain formulas. Given a state for R, program R assigns a set of integer number tuples $R(P)$ of proper arity for each intensional name P in R. $R(P)$ is defined step-by-step. Steps are enumerated by pairs of natural numbers. In steps (i,j), the mapping adds new tuples to $R(P)$ for intensional names P of the rank i.

The first step is $(1,0)$. Before the first step $R(P)$ are empty for all the intensional names.

For a step (i,j), $P(a_1, a_2, \ldots, a_n)$ is true iff either the tuple $a_1, a_2, \ldots, a_n$ was included in P before this step, or P is an extensional name and the tuple is contained in the set assigned to P by the state. $\neg P(a_1, a_2, \ldots, a_n)$ is true iff the rank of the intensional name P is less than i and $P(a_1, a_2, \ldots, a_n)$ is false.

A *rule instantiation* is defined as a substitution for each variable in the rule of an integer number. Let

$$P(a_1, a_2, \ldots, a_n) \longleftarrow A_1, A_2, \ldots, A_m$$

be a rule instantiation for a rule in R, P of the rank i, and let $A_1, A_2, \ldots, A_m$ be true for a step (i,j). Then the tuple $a_1, a_2, \ldots, a_n$ is called *marked* for P at the step (i,j) iff the tuple $a_1, a_2, \ldots, a_n$ was not included in P before this step.

In the step (i,j), if there is no intensional name of the rank i, the program R stops. Otherwise, for all P of the rank i and for all tuples $a_1, a_2, \ldots, a_n$ marked for P at the step, the tuple $a_1, a_2, \ldots, a_n$ is added to $R(P)$ at the step. If no tuple is added at step (i,j) we proceed to the step $(i+1,0)$. Otherwise we proceed to the step $(i,j+1)$.

A state is said to be *finite* for a program R iff the program stops in it. A program is finite or *safe* iff all states are finite for it.

Theorem 4.1. *Any Datalog program without negations is safe.*

Proof. Indeed, any input is a disjunction of conjunctions of expressions of one of the forms:

$$x <_p y, \quad x = y$$

where x and y are either a constant or a variable, and p is a natural number. If we existensionally quantify such a formula the result can be transformed in the same form. Thus, for a given program and state and for a fixed intensional name of the program, the program execution produces an increasing sequence of such formulas as the interpretations of the name. The program rules and the state formulas initially representing the extentional names use constants. A program execution cannot produce any new constant. Let A be the set of all the constants and all the variables using by the program name. The number of the variables is the arity of the name. Fix an ordering of A. For the ordering, any tuple of the values of the variables produces the tuple of the positive distances between the elements of A. Two tuples of the distances are compared by the following rule. A distance tuple is less than other one if any its distance is less than or is equal to the corresponding distance from the second distance tuple. A distance tuple $\bar{a}$ is minimal iff for any distance tuple $\bar{b}$, $\bar{a}$ is less than $\bar{b}$ implies that $\bar{a}$ is equal to $\bar{b}$. For any distance tuple $\bar{a}$ there is such a minimal distance tuple $\bar{b}$ that $\bar{b}$ is less than $\bar{a}$. It is easy to see that if a value tuple is contained in the interpretation of the name and a second value tuple produces a bigger or equal distance tuple then the second tuple is contained in the interpretation too. So for the ordering, the interpretation is defined by its set of all the minimal distance tuples. The interpretation of the name for the next step contains the interpretation for the previous one. So any minimal distance tuple for the next step or is incomparable with any minimal distance tuple for the previous one, or is less than a minimal distance tuple for the previous one, or is equal to a minimal distance tuple for the previous one.

Lemma 4.2. *The sequence of the sets of all the minimal distance tuples is finite.*

Lemma 4.2 arises from the following well known facts.

Fact 4.3. *Let k be a positive integer number and B be a set of k-tuples of positive integer numbers. The set of all the tuples minimal in B is finite.*

Here a k-tuple is a tuple of the length k.

Proof. Induction on k. For $k = 1$, it is trivial. Let $(a_1, a_2, \ldots, a_k) \in B$. Denote $B_i = \{(c_1, c_2, \ldots, c_k) \in B : \ c_i \leqslant a_i\}$. Let M_i be the set of all the tuples minimal in B_i. By the induction, M_i is finite for any i. For any tuple $\bar{b}$ minimal in B, there is i such that $b_i \leqslant a_i$. So $\bar{b} \in M_i$. ∎

Fact 4.4. *Let k be a positive integer number and $B_1, B_2, \ldots$ be a sequence of pairwise different sets of pairwise incomparable k-tuples of positive integer numbers. Suppose that any tuple from B_{i+1} either is incomparable with any tuple from B_i or is less than or equal to a tuple from B_i. Then the sequence is finite.*

Proof. By Fact 4.3, there is j such that for $i > j$, any tuple from B_{i+1} is less than or equal to a tuple from B_i. Thus the sequence is finite. ∎

So the sequence of the steps of the execution is finite. ∎

Two safe programs R_1 and R_2 with the same extensional names are equivalent in a state iff they have the same output intensional names of the same arities, and for each output intensional name P, $R_1(P)=R_2(P)$ in the state. Two safe programs are equivalent iff they are equivalent in any state.

4.2. **Impossibility of safe syntax.** The goal of this subsection is to prove that there is no effective syntax for safe programs. Let us outline the idea of the proof. Consider Turing machines in the alphabet $\{0,1\}$, where 0 is used as the blank symbol. A one-way infinite input tape for such a machine contains finitely many 1's in the first few positions, which can be interpreted as a natural number in the unary notation, and all other cells contain 0. If, in an input, such a machine stops, it leaves finitely many 1's on the tape. We may consider the first uninterrupted string of 1's left on the tape to be the output natural number in the unary notation. Then, every machine defines a partial function on natural numbers.

We want to show that, for any such Turing machine, there exists a stratified program that computes the same function. First problem is, programs do not work with tapes, they work with database states. Then, these states are finitely representable, but not necessarily finite. However, we can develop a coding scheme that will represent any natural number in the unary notation in the form of a *finite* database state. Perhaps the simplest such coding is by a unary predicate N as follows.

- if N is assigned a set of a cardinality 0 or > 2, it does not represent a number
- N assigned a set of cardinality 1 or 2 represents the natural number

$$\max(N) - \min(N)$$

For a natural number n, let $\hat{n}$ denote its representation in the form of unary predicate. Consider a stratified program R whose signature includes a single extensional predicate $INPUT$ and a single output intensional predicate $OUTPUT$. If, for a number n, R terminates when $\hat{n}$ is assigned to $INPUT$, $OUTPUT$ is assigned a certain set. If this set is $\hat{m}$ for some m, we say that $R(n) = m$. Otherwise we say that $R(n) = 0$. This way, every such program R defines a partial function on natural numbers.

Henceforth, we consider programs whose only extensional predicate is a unary $INPUT$, and only output intensional predicate is a unary $OUTPUT$. However, these programs may have other internal intensional predicates. The idea of our proof is to show that total Turing machines and *safe* stratified programs are effectively translatable to each other in the way that preserves the functions they define on natural numbers. It is known ([End72]) that total Turing machines do not have any effective syntax.

Theorem 4.5. *For any stratified program R there exists — and can be effectively constructed — a Turing machine M that defines the same partial function on natural numbers. If R is safe, the construction gives a total M.*

Proof. The target program works as follows. Given a number n, it assigns $\hat{n}$ to $INPUT$ and then interprets the computation by R step-by-step, according to the stepwise definition of semantics of a stratified program. At each step (i, j), our machine stores the values of all intensional predicates in the form of first-order pure domain formulas.[8] This computation may never end, and then the result in n is undefined. However, if R terminates in $\hat{n}$, our interpretation terminates too, and as a result, we have a value for $OUTPUT$ in the form of a first-order pure domain formula (note that, since the pure domain theory admits elimination of quantifiers, this representation can be translated into a finite representation, although we do not need this).

Since the pure domain theory is decidable, we can effectively determine whether this value for $OUTPUT$ represents a number. If it does, we can determine which number, and then write this number in the unary notation down to the tape and stop, otherwise, we write 0 in the unary notation to the tape and stop.

This computation can be carried out by a Turing machine, although explicitly writing such a machine would be a long boring exercise. Finally, if R is safe, it always terminates, and particularly it terminates when working on representations for natural numbers. Hence, for a safe R, the machine is total. ∎

The other direction is slightly more technical:

Theorem 4.6. *For any Turing machine M there exists — and can be effectively constructed — a stratified program R that defines the same partial function on natural numbers. If M is total, the construction gives a safe R.*

Proof. We want to concentrate on the case when the value assigned to $INPUT$ does represent a number. However, since we are going to construct a safe R, the case when it does not represent a number shall also be considered. It turns out that the program can decide whether $INPUT$ represents a number from the outset, and then if it does not, simply terminate right away with no matter which value for $OUTPUT$ — notice that as far as the numerical function goes it does not matter.

For example, the program may use nullary intensional predicates BAD and $GOOD$ to make this determination as follows:

$$\begin{aligned} GOOD &\longleftarrow INPUT(x), \neg BAD \\ BAD &\longleftarrow INPUT(x), INPUT(y), INPUT(z), \\ &\qquad x \neq y, x \neq z, y \neq z \end{aligned}$$

If $GOOD$ is true, it indicates that the database state does indeed represent a number. So all the other rules in our program may start with $GOOD$, and this guarantees termination for non-numerical inputs right away. We will omit $GOOD$ from the rules below, just to simplify notation.

[8] i.e., such formulas that only use constants and the integer (gap)-orders.

Further, we need to select some number to serve as 0. For a numerical state, we can pick up the minimal element in $INPUT$ as 0. Formally, this can be done by a stratified program that defines a unary predicate $ZERO$ to include this minimal number only, however, to simplify notation, we will simply use 0 as a constant. Similarly, we will use a constant max for the maximal number in $INPUT$, and constants $1, 2, \ldots, |Q|$, where Q is the set of the internal states of our Turing machine. Clearly, using any of these constants in the rules is simply an abbreviation for a long routine list of formulas.

We will also use a binary successor relation S, $S(x,y) \iff x+1=y$. This relation is definable using the gap orders as follows:

$$S(x,y) \longleftarrow x < y, \neg(x <_1 y)$$

To simulate computation by a Turing machine, we will use the following list of internal intensional predicates:

- ternary $TAPE$. $TAPE(i,j,k)$ indicates that in the step i of our computation the cell number j contains symbol k (0 or 1)
- binary $CELL$. $CELL(i,j)$ indicates that in the step i of our computation the cell number j is the current position of the machine
- binary $STATE$. $STATE(i,j)$ indicates that in the step i of our computation the internal state is j

The initial configuration of the Turing machine M can be explained by the following rules (we assume that the initial state is always 0):

$$\begin{aligned} TAPE(i,j,k) &\longleftarrow i=0, j \geqslant \max, k=0 \\ TAPE(i,j,k) &\longleftarrow i=0, j < \max, j \geqslant 0, k=1 \\ \\ CELL(i,j) &\longleftarrow i=0, j=0 \\ \\ STATE(i,j) &\longleftarrow i=0, j=0 \end{aligned}$$

We can also include the following rule asserting that the cells which are different from the current position of the machine do not change:

$$TAPE(i,j,k) \longleftarrow S(\ell,i), TAPE(\ell,j,k), CELL(\ell,c), c \neq j$$

Further simulation of the Turing machine is done according to the rules of this machine. Generally, a rule is of the form:

$$(q,k) \implies (s,m,n,a),$$

and it indicates that when the machine sees the symbol k in the internal state q, it replaces k in the current cell with s, moves according to $m \in \{\text{left}, \text{right}, \text{stay}\}$, and, generally, goes into the internal state n. If, however, the movement prescribed by the rule is left, but the current cell is the leftmost and no left movement is possible, the machine goes into the internal state a.

For each such Turing machine rule, we include a set of rules into our program. For example, let:

$$(3,1) \implies (0, \text{left}, 2, 7)$$

be a rule in our Turing machine. It causes inclusion of the following set of rules into our program:

$$\begin{aligned}
TAPE(i,j,k) &\longleftarrow S(\ell,i), TAPE(\ell,j,1), CELL(\ell,j),\\
&\qquad STATE(\ell,3), k=0
\end{aligned}$$

$$\begin{aligned}
CELL(i,j) &\longleftarrow S(\ell,i), TAPE(\ell,j,1), CELL(\ell,j),\\
&\qquad STATE(\ell,3), j=0\\
CELL(i,j) &\longleftarrow S(\ell,i), TAPE(\ell,j,1), CELL(\ell,x),\\
&\qquad STATE(\ell,3), S(j,x), x \neq 0
\end{aligned}$$

$$\begin{aligned}
STATE(i,j) &\longleftarrow S(\ell,i), TAPE(\ell,p,1), CELL(\ell,p),\\
&\qquad STATE(\ell,3), p=0, j=7\\
STATE(i,j) &\longleftarrow S(\ell,i), TAPE(\ell,p,1), CELL(\ell,p),\\
&\qquad STATE(\ell,3), p \neq 0, j=2
\end{aligned}$$

Let us now define unary predicates $LAST$ and MAX as follows:

$$\begin{aligned}
LAST(\ell) &\longleftarrow STATE(\ell,x), S(\ell,p),\\
&\qquad \neg STATE(p,0), \neg STATE(p,1), \ldots,\\
&\qquad \neg STATE(p,|Q|)\\
MAX(m) &\longleftarrow LAST(\ell), n<m, n \geqslant 0, TAPE(\ell,n,0)
\end{aligned}$$

Clearly, $LAST$ defines the last step in the computation by the Turing machine, if it stops. MAX defines the set of positive integer numbers m such that on the resulting tape left by our computation, there is at least one cell that is numbered lower than m and contains a 0. In particular, if the tape left by our Turing machine contains only 0's, the predicate is going to contain all positive integers.

We can finally define $OUTPUT$ as a predicate of the next rank as follows:

$$\begin{aligned}
OUTPUT(m) &\longleftarrow m=0\\
OUTPUT(m) &\longleftarrow m>0, \neg MAX(m), S(m,x), MAX(x)
\end{aligned}$$

Clearly, the value of $OUTPUT$ is going to be exactly the result of the computation by our Turing machine in our input, of course if this computation terminates. Since total Turing machines always terminate, our program R for a total Turing machine M is safe: given an $INPUT$ $\hat{n}$ representing n, it computes $\hat{m}$ such that $m = M(n)$, in a state that does not represent a natural number, it terminates right away. ∎

Corollary 4.7. *There is no effective syntax for safe programs.*

Proof. Indeed, the existence of such an effective syntax would, by Theorem 4.5, yield a recursive enumeration of total Turing machines such that every Turing computable total function is computed by a machine in this enumeration — here we use Theorem 4.6 to assure that every Turing computable total function appears in the enumeration. Such an enumeration is known to be impossible (see [Rog67]). ∎

References

[AGSS86] A.K. Ailamazian, M.M. Gilula, A.P. Stolboushkin, and G.F. Schwarz, *Reduction of the relational model with infinite domains to the case of finite domains*, Doklady Akademii nauk SSSR **286** (1986), no. 2, 308–311, Russian.

[AH91] A. Avron and J. Hirshfeld, *On first order database query languages*, Proc. 6th IEEE Symp. on Logic in Computer Science, 1991, pp. 226–231.

[AU79] A.V. Aho and J.D. Ullman, *Universality of data retrieval languages*, Proc. 6th ACM Symp. on Principles of Programming Languages, 1979, pp. 110–117.

[BDLW96] M. Benedikt, G. Dong, L. Libkin, and L. Wong, *Relational expressive power of constraint query languages*, Proc. 15th ACM Symp. on Principles of Database Systems, 1996, pp. 5–16.

[BL96] M. Benedikt and L. Libkin, *On the structure of queries in constraint query languages*, Proc. 11th IEEE Symp. on Logic in Computer Science (Los Alamitos, CA), IEEE Computer Society Press, 1996.

[BST96] O.V. Belegradek, A.P. Stolboushkin, and M.A. Taitslin, *On order-generic queries*, Technical report 96–01, DIMACS, 1996.

[BST97a] O.V. Belegradek, A.P. Stolboushkin, and M.A. Taitslin, *Extended order-generic queries*, Tver University, manuscript submitted to Annals of Pure and Applied Logic, 1997.

[BST97b] O.V. Belegradek, A.P. Stolboushkin, and M.A. Taitslin, *Generic queries over quasi-o-minimal domains*, LFCS'97, 1997.

[CH80] A. Chandra and D. Harel, *Computable queries for relational databases*, Journal of Computer and System Sciences **21** (1980), 156–178.

[CH82] A. Chandra and D. Harel, *Structure and complexity of relational queries*, Journal of Computer and System Sciences **25** (1982), 99–128.

[CK90] C.C. Chang and H.J. Keisler, *Model theory*, 3rd ed., North Holland, 1990.

[Cod70] E.F. Codd, *A relational model for large shared data banks*, Communications of the ACM **13** (1970), 377–387.

[Cod72] E.F. Codd, *Relational completeness of data base sublanguages*, Database Systems (R. Rustin, ed.), Prentice-Hall, 1972, pp. 33–64.

[Di 69] R.A. Di Paola, *The recursive unsolvability of the decision problem for the class of definite formulas*, Journal of the ACM **16** (1969), no. 2, 324–327.

[ELTT65] Yu.L. Ershov, I.A. Lavrov, A.D. Taimanov, and M.A. Taitslin, *Elementary theories*, Russian Mathematical Surveys **20** (1965), no. 4, 35–105.

[End72] H.B. Enderton, *A mathematical introduction to logic*, Academic Press, New York, 1972.

[GS94] S. Grumbach and J. Su, *Finitely representable databases*, Proc. 13th ACM Symp. on Principles of Database Systems, 1994.

[GS95] S. Grumbach and J. Su, *Dense-order constraint databases*, Proc. 14th ACM Symp. on Principles of Database Systems, 1995, pp. 66–77.

[GSSS86] M.M. Gilula, A.P. Stolboushkin, G.F. Schwarz, and K.V. Shvachko, *Some algorithmic problems in the theory of relational databases*, Problem-Oriented Computational Systems (Moscow, USSR) (A.K. Ailamazian, ed.), Problems in Cybernetics, vol. 125, USSR Academy of Sciences, Moscow, USSR, 1986, Russian, pp. 81–91.

[GST95] S. Grumbach, J. Su, and C. Tollu, *Linear constraint databases*, Proc. Logic and Computational Complexity (LCC'94), LNCS, Springer-Verlag, 1995.

[GT91] S. Grumbach and C. Tollu, *The generic complexity of query languages with counters*, Proceedings of 3rd Workshop on Foundations of Models and Languages for Data and Objects,Aigen (Austria), 1991.

[Gur88] Y. Gurevich, *Logic and the challenge of computer science*, Current trends in theoretical computer science (E. Börger, ed.), Computer Science Press, 1988, pp. 1–57.

[Gur90] Yu. Gurevich, Private communication, 1990.

[Hir91] J. Hirshfeld, *Safe queries in relational databases with functions*, CSL '91: 5th Workshop on Computer Science Logic, LNCS, Springer-Verlag, 1991, pp. 173–183.

[JL87] J. Jaffar and J.-L. Lassez, *Constraint logic programming*, Proc. 14th ACM Symp. on Principles of Programming Languages, 1987, pp. 111–119.

[JM94] J. Jaffar and M.J. Maher, *Constraint logic programming: A survey*, Journal of Logic Programming **19–20** (1994), 503–581.

[Kan88] P.C. Kanellakis, *Logic programming and parallel complexity*, Foundations of Deductive Databases and Logic Programming (J. Minker, ed.), Morgan Kaufmann, 1988, pp. 547–586.

[KG94] P.C. Kanellakis and D.Q. Goldin, *Constraint programming and database query languages*, Proc. International Symposium on Theoretical Aspects of Computer Software (TACS'94), 1994, pp. 96–120.

[Kif88] M. Kifer, *On safety, domain independence, and capturability of database queries*, Proc. Third International Conference on Data and Knowledge Bases, 1988.

[KKR90] P.C. Kanellakis, G.M. Kuper, and P.Z. Revesz, *Constraint query languages*, Proc. 9th ACM Symp. on Principles of Database Systems, 1990, pp. 299–313.

[KKR95] P.C. Kanellakis, G.M. Kuper, and P.Z. Revesz, *Constraint query languages*, Journal of Computer and System Sciences **51** (1995), no. 1, 26–52.

[KPS86] J.F. Knight, A. Pillay, and C. Steinhorn, *Definable sets in ordered structures, II*, Transactions of American Mathematical Society **295** (1986), no. 2, 593–605.

[OV95] M. Otto and J. Van den Bussche, *First-order queries on databases embedded in an infinite structure*, Manuscript, 1995.

[PS86] A. Pillay and C. Steinhorn, *Definable sets in ordered structures, I*, Transactions of American Mathematical Society **295** (1986), no. 2, 565–592.

[PS88] A. Pillay and C. Steinhorn, *Definable sets in ordered structures, III*, Transactions of American Mathematical Society **309** (1988), no. 2, 469–476.

[PVV95] J. Paradaens, J. Van den Bussche, and D. Van Gucht, *First-order queries on finite structures over reals*, Proc. 10th IEEE Symp. on Logic in Computer Science, IEEE Computer Society Press, 1995, pp. 79–87.

[Rab77] M.O. Rabin, *Decidable theories*, Handbook of Mathematical Logic (Amsterdam, New York, Oxford) (J. Barwise, ed.), vol. 3, North-Holland, Amsterdam, New York, Oxford, 1977.

[Rev90] P.Z. Revesz, *A closed form for Datalog queries with integer order*, 3rd Int'l. Conf. on Database Theory, Springer-Verlag, 1990, pp. 187–201.

[Rev93] P.Z. Revesz, *A closed form evaluation for Datalog queries with integer (gap)-order constraints*, Theoretical Computer Science **116** (1993), no. 1, 117–149.

[Rev95] P.Z. Revesz, *Safe stratified Datalog with integer order programs*, Manuscript, August 1995.

[Rob49] Julia Robinson, *Definability and decision problems in arithmetic*, Journal of Symbolic Logic **14** (1949), 98–114.

[Rog67] H. Rogers, *Theory of recursive functions and effective computability*, McGaw-Hill, 1967.

[RW81] B.I. Rose and R.E. Woodrow, *Ultrahomogeneous structures*, Zeitschrift für Mathematische Logik und Grundlagen der Mathematik **27** (1981), no. 1, 23–30.

[Sac72] G.E. Sacks, *Saturated model theory*, W.A. Benjammin, Inc., Reading, Massachusetts, 1972.

[ST95a] A.P. Stolboushkin and M.A. Taitslin, *Finite queries do not have effective syntax*, Proc. 14th ACM Symp. on Principles of Database Systems, 1995, pp. 277–285.

[ST95b] A.P. Stolboushkin and M.A. Taitslin, *Is first order contained in an initial segment of PTIME?*, Selected Papers, 8th EATCS Conference on Computer Science Logic (CSL'94), LNCS, vol. 933, Springer-Verlag, 1995, pp. 242–248.

[ST95c] A.P. Stolboushkin and M.A. Taitslin, *Safe stratified datalog with integer order does not have syntax*, Manuscript, October 1995.

[ST96] A.P. Stolboushkin and M.A. Taitslin, *Linear vs. order constraint queries over rational databases*, Proc. 15th ACM Symp. on Principles of Database Systems, 1996, pp. 17–27.

[Ull82] J.D. Ullman, *Principles of database systems*, 2 ed., Computer Science Press, 1982.

[Ull88] J.D. Ullman, *Principles of database and knowledge-base systems, volumes i and ii*, Computer Science Press, 1988.

[Van91] Van Gelder, A. and R.W. Topor, *Safety and translation of relational calculus queries*, ACM Trans. on Database Systems **16** (1991), no. 2, 235–278.

[Var81] M.Y. Vardi, *The decision problem for database dependencies*, Information Processing Letters (1981), 251–254.

O.V.Belegradek, Department of Mathematics, Kemerovo State University, Kemerovo, Russia 650043

E-mail address: `beleg@kaskad.uucp.stanet.ru`

Alexei P. Stolboushkin, Fourth Dimension Software, 555 Twin Dolphin Dr., Redwood City, CA 94404, and UCLA Mathematics Department, Los Angeles, CA 90095

E-mail address: `aps@4ds.com`

E-mail address: `aps@math.ucla.edu`

M.A.Taitslin, Department of Computer Science, Tver State University, Tver, Russia 170000

E-mail address: `mat@tversu.ac.ru`