Queries with Arithmetical Constraints ⋆

Stéphane Grumbach [a,1] and Jianwen Su [b,2]

[a] *I.N.R.I.A., Rocquencourt BP 105, 78153 Le Chesnay, France, E-mail: stephane.grumbach@inria.fr*

[b] *Department of Computer Science, University of California, Santa Barbara, CA 93106, USA, E-mail: su@cs.ucsb.edu*

This paper is dedicated to Paris Kanellakis who initiated the field of constraint databases, and deeply influenced our work.

Abstract

In this paper, we study the expressive power and the complexity of first-order logic with arithmetic, as a query language over relational and constraint databases. We consider constraints over various domains ($\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, and $\mathbb{R}$), and with various arithmetical operations ($\leqslant$, $+$, $\times$, etc.).

We first consider the data complexity of first-order queries. We prove in particular that linear queries can be evaluated in AC^0 over finite integer databases, and in NC^1 over linear constraint databases. This improves previously known bounds. We also show that over all domains, enough arithmetic lead to arithmetical queries, therefore, showing the frontiers of constraints for database purposes.

We then tackle the problem of the expressive power, with the definability of the parity and the connectivity, which are the most classical examples of queries not expressible in first-order logic over finite structures. We prove that these two queries are first-order definable in presence of (enough) arithmetic. Nevertheless, we show that they are not definable with constraints of interest for constraint databases such as linear constraints for instance. Finally, we developed reduction techniques for queries over constraint databases, that allow us to draw conclusions with respect to their undefinability in various constraint query languages.

⋆ An extended abstract of the paper appeared in *Proc. Int. Conf. on Principles and Practice of Constraint Programming (CP95), Sept. 1995.*

[1] Work supported in part by an NSERC fellowship in Canada.

[2] Work supported in part by NSF grants IRI-9117094, IRI-9411330 and NASA grant NAGW-3888.

1 Introduction

The expressive power of first-order logic has been studied in classical logic [CK73], where powerful tools, such as the compactness theorem, Ehrenfeucht-Fraïssé games, etc., provide easy answers to the definability of numerous problems. More recently, a variant of the general problem has been considered with restrictions on the semantics. The definability over the class of finite structures has attracted a considerable attention in finite model theory [Fag93]. The problem differs drastically from the initial one, since most of the powerful tools of classical logic fail when the semantics is restricted to classes of models, such as finite models for instance. New techniques have been developed such as the study of the asymptotic probabilities of the truth of sentences [Fag76].

Finite model theory is strongly motivated by database theory. A relational database is just a finite structure. The new applications of database systems (e.g. geographical information) have lead to new data models, appropriate for the manipulation of geometrical, topological, and arithmetical information. Constraints over arithmetical domains provide a excellent framework for such data models. Their introduction into first-order logic raises questions relative to the expressive power of first-order queries, over both classical finite databases, and newly defined constraint databases, that constitute the topic of the paper.

Until recently, databases were considered to be finite collections of data items. New applications such as those involving temporal and spatial data lead very naturally to more general data models allowing infinite collections of items to be stored in the database. The *constraint databases,* introduced by Kanellakis, Kuper, and Revesz in their seminal paper [KKR90], constitute a powerful generalization of Codd's relational model. In this new paradigm, instead of tuples, queries act on "generalized tuples" expressed as quantifier-free first-order constraints in a decidable theory. A generalized relation is a finite conjunction of such constraints, interpreted in the domain of a given model of the decidable theory. Interesting constraint query languages are then obtained by coupling the relational calculus (or some version of Datalog) with a decidable theory.

Different types of constraints were considered in [KKR90] such as dense linear order inequalities [KG94], real polynomial inequalities, etc. These constraints are based on decidable theories that admit the elimination of quantifiers. The data complexity of both the relational calculus and inflationary Datalog [Rev93,Rev95] with negation over dense order constraint databases has been shown to be tractable, though the decision problem of the underlying theories might have high complexity. The low data complexity shows the promising potential for practical applications of this approach, which has been pursued in particular in [Kup93,PVV94,CK95,GS95,PVV95].

We consider constraints over the domains of the natural numbers, the integers, the rationals and the reals, expressed in first-order languages with equality, and possibly an order relation and arithmetic operations, such as addition, multiplication, exponentiation (e^x), super-exponentiation (x^y), etc.

We investigate the data complexity, which is the complexity of the evaluation of queries in terms of the database size. We first consider the complexity over finite relations (classical relational case). We prove that linear queries (expressed with order and addition) can be evaluated in AC^0 data complexity, that is on Boolean circuits with arbitrary fan-in gates with constant parallel time and a polynomial number of processors. This result shows that linear queries have the same potential for parallelization than first-order logic with no arithmetic. On the other hand, we proved that while polynomial queries over the reals can be evaluated in NC [BKR86,KKR90], polynomial queries over the natural numbers, the integers and the rationals are arithmetical.

We then consider the complexity over finitely representable relations and prove a new result for the case of linear queries that, over all the domains considered, they can be evaluated in NC^1 (logarithmic time with a polynomial number of processors over bounded fan-in Boolean circuits). The proof is based on the following observation. The computation of a linear query requires the use of a bounded number of nested integer multiplications, which is known to be computable within logarithmic parallel time [SS71]. Order queries have AC^0 bounds while polynomial queries admit the same bound as over finite structures.

The data complexity of first-order logic provides an upper-bound on the expressive power. We investigate in detail the impact of the arithmetic. The PARITY of the cardinality of a set and the CONNECTIVITY of a finite graph are well-known queries that are not definable over finite models in a first-order language with equality. They are especially interesting, since they involve two basic primitives: counting and recursion. Their undefinability was shown by various proof techniques such as locality [Gai81], asymptotic probabilities [Fag74], Ehrenfeucht-Fraïssé games [Fra54,Ehr61], etc. The undefinability results also hold in the presence of an order relation on the domain [Gur88].

It was shown in [GS94] that the model theory of finitely representable structures, differs strongly from the classical model theory of all structures. In particular, most of the classical results of logic such as the compactness and the completeness theorems fail for finitely representable structures, like for finite models (see the survey by Fagin [Fag93]). Constraint databases thus introduce a new area of research on first-order definability. The previous undefinability results were shown for domains allowing at most an order relation. It is particularly challenging that the existing techniques do not carry over in presence of arithmetic. Ehrenfeucht-Fraïssé games apply, but they lead to inextricable combinatorics [GS94], while locality and 0/1 laws do not work already in the presence of an order relation.

On the other hand, the PARITY and CONNECTIVITY queries have nothing to do with arithmetic. They are *generic* [CH80], i.e., they commute with permutations of the domain, even permutations violating the order and the arithmetic operations. The increase of the number of relations and arithmetic functions in a first-order language generally results in an increase of its expressive power. This is clear for all classical arithmetics over the natural numbers or the reals for instance with the operations $+$ and $\times$.

Nevertheless, it is a priori not clear that more generic queries are expressible in a language extended with arithmetic operations. Gurevich ([AHV95], Exercise 17.27, page 162) found an example of a generic query expressible with an order relation, and not expressible without it. We prove here that PARITY and CONNECTIVITY are definable over the natural numbers or the rationals with addition and multiplication, and over the reals with super-exponentiation. On the other hand these queries are not definable with addition only. The expressive power of linear queries has also been studied in [ACGK94]. Recently, it was shown in [BDLW95] that PARITY was not definable with polynomial constraints over the reals.

We develop reduction techniques to prove the undefinability of numerous queries under the assumption that parity is not definable. We illustrate these techniques on queries from geometry, topology, computational geometry [Yao90], graph theory, and geographical databases, and we investigate their first-order definability in various contexts. Our reduction results and the result of [BDLW95] answer most of the definability questions over constraint databases found in the literature.

The paper is organized as follows. In the next section, we present the concept of relational databases and constraint databases. In Section 3, we introduce queries and first-order logic as a query language, and study the data complexity of first-order queries for various domains, arithmetic, and over finite or finitely representable relations. Section 4 is devoted to relational generic queries, while queries over finitely representable queries are studied in Section 5, with the reduction techniques.

2 From Finite to Constraint Databases

We here introduce the framework of *finitely representable* databases which extends the classical paradigm of the relational model, and present a few important examples of such databases.

All the results presented here are related to first-order logic, which deserves therefore a brief presentation. In the whole paper, we consider *first-order languages* with equality. A first-order language $\mathcal{L}$ consists of *predicate*, *function*, and *constant* symbols. The family of *terms* is recursively defined as follows: a constant or a variable is a term, $f(t_1, ..., t_n)$ is a term if f is an n-ary function symbol and $t_1, ..., t_n$ are terms. Formulas are defined using logical connectives, terms, and predicate symbols from $\mathcal{L}$ in the standard way: $P(t_1, ..., t_n)$ is an *atomic formula* if P is an n-ary predicate symbol and $t_1, ..., t_n$ are terms; $\neg\varphi$, $\varphi \vee \psi$, $\varphi \wedge \psi$, $\exists x\, \varphi$, and $\forall x\, \varphi$ are formulas if φ, ψ are formulas and x is a variable. A *sentence* is a formula with no free occurrences of variables. A formula is *quantifier-free* if no quantifier occurs in it.

A *structure* for a first-order language $\mathcal{L}$ (an $\mathcal{L}$*-structure*) is a pair $\mathcal{A} = \langle A, \rho\rangle$, where A is a non empty set, called the *universe*, and ρ is the interpretation function mapping predicate, function, and constant symbols in $\mathcal{L}$ to appropriate relations over, functions on, and elements in A, respectively. The notion of *satisfaction* is defined in the standard way. We say equivalently that a structure $\mathcal{A}$ is a *model* of a sentence φ, or that φ is *true* in $\mathcal{A}$, and denote it by $\mathcal{A} \models \varphi$. The set of sentences true in a structure $\mathcal{A}$ is called the *theory of* $\mathcal{A}$. A theory $\mathcal{T}$ (set of sentences closed under logical implication) is complete, whenever for each sentence φ, we have $\varphi \in \mathcal{T}$ or $\neg\varphi \in \mathcal{T}$.

In this paper, we shall focus on the following numerical domains:

- $\mathbb{N}$ of natural numbers,
- $\mathbb{Z}$ of integers,
- $\mathbb{Q}$ of rational numbers, and
- $\mathbb{R}$ of real numbers

along with the (total) order predicate ($\leqslant$) and the following arithmetic operations: addition ($+$), multiplication ($\times$), exponentiation (e^y) and super-exponentiation (x^y). The super-exponentiation function x^y is defined over the complex numbers $\mathbb{C}$, and is only partially defined over the real numbers $\mathbb{R}$ and the rational numbers $\mathbb{Q}$ (if x is negative, x^y may not be defined for some y's). First-order logic with arithmetic is treated in great detail in [End72], which is of great help for the present topic.

We now review the well known framework of finite relational databases [Cod70,Ull88]. The universe is restricted to the natural numbers, possibly with the order, but without arithmetic.

A *database schema* σ is a set of relation symbols such that $\mathcal{L} \cap \sigma = \varnothing$. Throughout the paper, we assume that the database schema is disjoint from the first-order language, and we distinguish between *logical predicates* (e.g., $=, \leqslant$) in $\mathcal{L}$ and *relations* in σ. A *database instance* is a finite structure, with a finite domain D, and a finite interpretation of the relation symbols.

The theoretical framework of relational databases is therefore restricted to finite models in contrast to all models in classical logic. The formal foundations have been studied in an insufficiently known branch of logic, finite model theory [Fag93], which is the model theory of finite structures. This field has attracted a recent and large development mainly due to its connection to database theory [AHV95]. Finite model theory differs drastically form classical model theory. Indeed, the two fundamental theorems of classical logic, the compactness and the completeness theorems, fail in the context of finite models. Various other results also fail such as the recursive enumerability of the set of sentences valid over finite models [Tra50], or Beth's theorem on the equivalence between implicit and explicit definitions [Gur88]. These results are fundamental to prove in particular expressive power results, and explain the need to develop alternative proof methods in finite model theory. We shall see that the situation is even worse for constraint databases.

We now consider constraint databases. Kanellakis, Kuper, and Revesz [KKR90] introduced the concept of *k-ary generalized tuple*, which is a constraint expressed as a conjunction of atomic formulas in $\mathcal{L}$ over k variables. A k-ary *finitely representable relation* (or generalized relation in [KKR90]) is then a finite set of k-ary generalized tuples. In this framework, a tuple $[a, b]$ in the context of classical relational databases [Cod70] is an abbreviation for the formula $(x = a \wedge y = b)$ represented using only the equality symbol, "=", and constants. Geometric objects can easily be defined. A triangle can be defined as a generalized tuple represented using only "$\leqslant$" and constants, by an expression of the form: $(x \leqslant y \wedge x \geqslant 0 \wedge y \leqslant 10)$. More sophisticated figures can be defined, such as a half circle over the reals for instance: $(x \times x + y \times y = 1 \wedge x \leqslant 0)$.

The structures are of particular interest in the context of finitely representable models include the discrete structures such as

- $\langle \mathbb{N}, \leqslant, (n)_{n \in \mathbb{N}} \rangle$, $\langle \mathbb{Z}, \leqslant, (n)_{n \in \mathbb{Z}} \rangle$,
- $\langle \mathbb{N}, \leqslant, +, (n)_{n \in \mathbb{N}} \rangle$, $\langle \mathbb{Z}, \leqslant, +, (n)_{n \in \mathbb{Z}} \rangle$,

over the languages of (respectively) order, order and addition, and natural or integer constants; and dense structures such as

- $\mathcal{Q} = \langle \mathbb{Q}, =, \leqslant, (q)_{q \in \mathbb{Q}} \rangle$, $\langle \mathbb{Q}, =, \leqslant, +, (q)_{q \in \mathbb{Q}} \rangle$,
- $\mathcal{R} = \langle \mathbb{R}, =, \leqslant, +, \times, (q)_{q \in \mathbb{A}} \rangle$,

over (respectively) the languages $\mathcal{L}_{\leqslant} = \{=, \leqslant\} \cup \mathbb{Q}$, and $\mathcal{L}_{\times} = \{=, \leqslant, \times, +\} \cup \mathbb{A}$, where $\mathbb{A}$ is the set of algebraic numbers.

We assume that the languages contain constants, either all constants in an enumerable domain ($\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$) or the set of algebraic numbers $\mathbb{A}$ in the case of the reals. In the sequel we will omit the constants in the notation of structures.

Other structures will be considered, such as the structure of number theory, $\langle \mathbb{N}, \leqslant, +, \times \rangle$, and structures over larger vocabulary, but we will see that they are generally meaningless for constraint databases. Note that all theories of the previous structures are decidable and have hyper-exponential complexity [FR79,Ren92].

The structure $\mathcal{Q}$ satisfies the theory of dense order without endpoints that is known to be complete [CK73]. Moreover, it admits the elimination of quantifiers, which as we shall see in the following is the *fundamental property* of constraint databases. The structure $\mathcal{R}$ satisfies the theory of ordered real closed fields which, like the theory of dense order without endpoints, is also complete and admits the elimination of quantifiers [CK73].

A constraint database is a finite representation of a structure, which is an *expansion* of some arithmetical structure, say $\mathcal{R}$, to a database schema σ, i.e., a structure over the vocabulary $\{=, \leqslant, +, \times, (q)_{q \in \mathbb{A}}\} \cup \sigma$ that coincides with $\mathcal{R}$ on $\{=, \leqslant, +, \times, (q)_{q \in \mathbb{A}}\}$. The new relations of σ constitute the database (in the *context* of $\mathcal{R}$). Although the relations may be infinite, they are *finitely* representable with symbols in $\{=, \leqslant, +, \times, (q)_{q \in \mathbb{A}}\}$.

Definition 2.1 Let $\mathcal{A}$ be an $\mathcal{L}$-structure with universe A, $\varphi(x_1, ..., x_n)$ a quantifier-free formula in $\mathcal{L}$ with n distinct free variables $x_1, ..., x_n$. An n-ary relation $S \subseteq A^n$ is *represented by φ over $\mathcal{A}$* if the following holds:

$$\text{for all } a_1, ..., a_n \in A, \quad \mathcal{A} \models \varphi(a_1, ..., a_n) \quad \textit{iff} \quad (a_1, ..., a_n) \in S \ .$$

If S is represented by φ over $\mathcal{A}$, φ is also called a *finite representation of S over $\mathcal{A}$*.

Example 2.2 Let a, b, c, d be rational numbers such that $a < c$ and $b < d$. In the rational plane ($\mathbb{Q}^2$), a rectangle with lower left and upper right corners respectively (a, b) and (c, d) is represented by the formula in $\mathcal{L}_{\leqslant}$ over $\mathcal{Q}$:

$$(a \leqslant x) \wedge (x \leqslant c) \wedge (b \leqslant y) \wedge (y \leqslant d).$$

In the 3-dimensional real space ($\mathbb{R}^3$), the top half of the sphere of distance d from the point (a, b, c) is defined by the following formula in $\mathcal{L}_{\times}$:

$$(x-a)^2 + (y-b)^2 + (z-c)^2 = d^2. \qquad \blacksquare$$

Finitely representable relations are somehow the exception in the universe of all relations, which is not enumerable in general. The binary relation defined by *cosine* for instance is not finitely representable in $\mathcal{L}_{\times}$ [Dri82].

Definition 2.3 Let $\mathcal{A}$ be an $\mathcal{L}$-structure with universe A. An n-ary relation $S \subseteq A^n$ is *finitely representable in $\mathcal{L}$* (*$\mathcal{L}$-representable* for short) *over $\mathcal{A}$* ($\mathcal{A}$ is omitted when it is clear from the context) if it is represented by a formula in $\mathcal{L}$ over $\mathcal{A}$. An expansion $\mathcal{B}$ of $\mathcal{A}$ to the schema σ is said to be *$\mathcal{L}$-representable (over $\mathcal{A}$)* if for every relation symbol R in σ, $R^{\mathcal{B}}$ is $\mathcal{L}$-representable (over $\mathcal{A}$).

Note that the languages considered contain a constant symbol for each rational number in the case of $\mathcal{Q}$, and for each algebraic number in the case of $\mathcal{R}$. This is necessary in our framework (algebraic numbers are definable with $+, \times, 0$, and 1, which may require additional variables and quantifiers).

Definition 2.4 An $(\mathcal{A}, \sigma)$-*(database) instance* is a mapping I such that there exists an expansion $\mathcal{B}$ of $\mathcal{A}$ to σ that is $\mathcal{L}$-representable over $\mathcal{A}$, and for each $R \in \sigma$, $R^{\mathcal{B}}$ is represented by $I(R)$ over $\mathcal{A}$.

When everything is clear from the context, we will simply speak of an instance. In the following, we fix $\mathcal{A}$ and σ, and consider the class $K(\mathcal{A}, \sigma)$ of $(\mathcal{A}, \sigma)$-instances. If $\mathcal{L}$ is a countable language, then $K(\mathcal{A}, \sigma)$ is effectively enumerable. Moreover, if $\mathcal{A}$ has a countable universe such as the natural numbers, then every relation in each $(\mathcal{A}, \sigma)$-instance in $K(\mathcal{A}, \sigma)$ is recursive. Note that $K(\mathcal{A}, \sigma)$ is closed under finite union, intersection, and complement. This differs from finite model theory (the complement of a finite model is not finite).

For each sentence φ in $\mathcal{L} \cup \sigma$, we denote by $K_\varphi(\mathcal{A}, \sigma)$ the collection of $(\mathcal{A}, \sigma)$-instances satisfying φ, i.e., for each $\mathcal{L}$-representable $(\mathcal{A}, \sigma)$-expansion $\mathcal{B}$, if $\mathcal{B} \models \varphi$, then each finite representation of $\mathcal{B}|_\sigma$ is in $K_\varphi(\mathcal{A}, \sigma)$. (We may denote $K_\varphi(\mathcal{A}, \sigma)$ simply by $K_\varphi^{\mathcal{A}}$ when σ is clear from the context.)

Database instances have a syntactic and a semantic aspect. We are merely interested by the semantics of the instances. We next define a notion of semantic equivalence of two instances.

Definition 2.5 Suppose $\mathcal{A}$ is an $\mathcal{L}$ structure. Let σ be a database schema. Two $(\mathcal{A}, \sigma)$-databases I and J are equivalent if for each $R \in \sigma$ of arity n,

$$\mathcal{A} \models \forall x_1 \cdots \forall x_n (I(R) \leftrightarrow J(R)).$$

A finitely representable relation can therefore be infinite, of arbitrary cardinality (not necessarily countable), but it admits a finite representation. A finite relation (database) is representable using only the equality predicate (and constants), and therefore is an $\{=\}$-representable relation (and the representing formula does not involve negation). The converse doesn't hold. However, a monadic relation is $\{=\}$-representable iff it is finite or co-finite.

3 Queries, query languages and data complexity

In this section, we define the fundamental concept of a "query" and introduce various subclasses of queries including the classical "generic" (relational) queries [CH80,CH82], dense-order queries [KKR90,GS95], and linear queries [GST94,PVV95]. We illustrate various concepts with examples queries from various fields, including geometry and graph theory. We then consider first-order languages as "query languages" and study the complexity bounds of evaluating queries expressed in these languages as a function of the input database size.

3.1 Queries

Queries are mappings which satisfy a consistency criterion called *genericity*, which reflects the independence of the data from its representation. In (finite) relational databases, a query, introduced by Chandra and Harel [CH80], is a mapping Q from finite structures over a given schema σ to relations of a fixed arity n satisfying the following conditions: (i) Q is partial recursive; and (ii) for each database I of σ and each permutation ρ of the underlying domain, $Q(\rho(I)) = \rho(Q(I))$, i.e., Q commutes with ρ. Condition (ii) is called *genericity* in the database literature. Note that the underlying structure of a relational database is a first-order structure $\mathcal{A}$ for the language $\mathcal{L}$ consisting of one predicate symbol (the equality "=") and no function symbols. Hence every permutation of the domain is an automorphism of $\mathcal{A}$. The genericity implies in particular that if two databases, I and J, over σ are isomorphic by an isomorphism, then $Q(I)$ and $Q(J)$ are also isomorphic by the same isomorphism.

Genericity was later generalized to "C-genericity" by Hull [Hul86] to allow the use of a fixed set C of constants (which are left fixed by the isomorphisms) in a query expression. In this paper, we only consider the genericity notion in its original form, though the results are generalizable. Constraint databases give raise to numerous concepts of genericity [PVV94,BDLW95]. Genericity is not the purpose of the present paper, and we adopt the following definition.

Definition 3.1 Let $\mathcal{A}$ be an $\mathcal{L}$-structure, σ a database schema, and $k \in \mathbb{N}$. A (k-ary) query is a mapping Q from $(\mathcal{A}, \sigma)$-databases to quantifier-free formulas in $\mathcal{L}$ with k distinct free variables $y_1, ..., y_k$ satisfying the following condition:

- For each pair I, J of databases over σ, if I and J are equivalent, then $Q(I)$ and $Q(J)$ are equivalent.

A Boolean query is a collection of $(\mathcal{A}, \sigma)$-databases closed under equivalence.

In the above definition, we do not require a query to be computable (or recursive), since we shall also consider query languages that are capable of expressing queries in the arithmetical hierarchy.

Definition 3.2 If $\mathcal{A}$ is an $\mathcal{L}$-structure with universe A and σ a database schema, then a query Q is *(relational) generic* if Q commutes with each permutation of A, i.e., for each $(\mathcal{A}, \sigma)$-database I and each permutation ρ of A, $Q(\rho(I)) = \rho(Q(I))$.

Generic queries make sense on finite inputs only. It can be verified that for a language $\mathcal{L} \subseteq \{\leqslant, +, \times\}$, if S is an $\mathcal{L}$-representable set over $\mathbb{Q}$ (or $\mathbb{R}$), and if for each permutation ρ of $\mathbb{Q}$ ($\mathbb{R}$), $\rho(S)$ is also $\mathcal{L}$-representable, then S is finite. This holds because if S is not finite, S must contain an interval since S is $\mathcal{L}$-representable. Now under permutations μ which break the interval into infinitely many "pieces", $\mu(S)$ is not finitely representable.

In the general context of finitely representable databases, e.g., over the real field $\mathcal{R}$, clearly the identity becomes the only isomorphism between the finitely representable extensions of $\mathcal{R}$. Recently, different notions of genericity have been proposed and studied in [KG94,PVV94,GS95] and we consider one of these in the following.

Definition 3.3 Suppose $\mathcal{L}$ is a first-order language with the order predicate $\leqslant$, $\mathcal{A}$ an $\mathcal{L}$-structure, and σ a database schema. A query Q is *$\leqslant$-generic* (*order generic*) if Q commutes with each isomorphism of the substructure $\mathcal{A}|_{\{\leqslant\}}$, i.e., the structure containing the universe of $\mathcal{A}$ and the binary relation $\leqslant^{\mathcal{A}}$.

We now present numerous examples of constraint database queries in the areas of geometry, topology, computational geometry, graph theory, and geographical databases. In Section 5, we exhibit reductions among these queries and the reduction results establish a hierarchy of equivalence classes of queries.

We consider mappings corresponding to (i) well known (Boolean) graph queries (parity, majority, etc.), and (ii) spatial queries (e.g., region connectivity). For graph queries, the inputs are finite relations (over, e.g., integer values) defined with equality constraints. For spatial queries, we consider queries over potentially infinite relations. For example, suppose that R is an infinite binary relation. It can be seen as defining an infinite set of points on the real plane. The queries considered here concern topological or geometric properties of the input relations.

More specifically, we consider the following graph queries, which are generic queries. For each finite relation R, $|R|$ denotes the cardinality of R.

- PARITY, **input**: a finite relation R; **output**: *true* iff $|R|$ is even.
- MAJORITY, **input**: two finite relations, R_1 and R_2, of the same arity; **output**: *true* iff $R_1 \subseteq R_2$, and $|R_2| \leqslant 2|R_1|$.
- HALF, **input**: two finite relations, R_1 and R_2, of the same arity; **output**: *true* iff $R_1 \subseteq R_2$, and $|R_2| = 2|R_1|$.
- CONNECTIVITY, **input**: a finite binary relation R representing a graph; **output**: *true* iff the graph represented by R is connected (i.e., there is a path from every node u to every node v).

The topological and geometric queries considered include the following $\leqslant$-generic queries.

- *k-dimensional* REGION CONNECTIVITY, **input**: one relation R of arity k; **output**: *true* iff R is connected, i.e., every pair of points in R can be linked by a curve contained entirely in R.
- *k-dimensional* AT LEAST (EXACTLY) ONE HOLE, **input**: one relation R of arity k; **output**: *true* iff R has at least (exactly) one hole (a connected region with empty intersection with R, but lying inside a hyper-sphere of dimension $k-1$ which is included in R). finite volume).
- EULERIAN TRAVERSAL, **input**: one binary relation R consisting of only line segments; **output**: *true* iff there is a traversal going through each line segment exactly once.
- HOMEOMORPHISM, **input**: two binary relations, R_1 and R_2; **output**: *true* iff the sets of points in R_1 and R_2 are homeomorphic in dimension 2.

Note that for $k = 1$, the REGION CONNECTIVITY, AT LEAST and EXACTLY ONE HOLE queries can be easily expressed in first order. We also consider the following query, EUCLIDEAN SPANNING TREE, which is not $\leqslant$-generic but was conjectured not expressible in first order in [KKR90] in the context of $\mathcal{R}$.

- EUCLIDEAN SPANNING TREE, **input**: one binary relation representing a finite set of points in $\mathbb{R}^2$; **output**: one relation with arity four representing a set of pairs of points which are edges of the Euclidean spanning tree.

3.2 Query languages

Let $\mathcal{L}$ be a first-order language and σ a database schema, disjoint from $\mathcal{L}$. We consider $\mathcal{L}$ as a query language: each formula φ in $\mathcal{L} \cup \sigma$ with free variables in $\{x_1, ..., x_n\}$ naturally defines a *query (expression) over* σ ($n \geqslant 0$):

$$\{(x_1, ..., x_n) \mid \varphi\}.$$

Suppose $Q = \{(x_1, ..., x_n) \mid \varphi\}$ is a query over the schema σ, and I is an $(\mathcal{A}, \sigma)$-database. Since for each $R \in \sigma$, $I(R)$ is a (quantifier-free) formula in $\mathcal{L}$, and φ is a formula in $\mathcal{L} \cup \sigma$, we can replace in φ each occurrence of a relation symbol R by the formula $I(R)$. Clearly, the resulting formula, denoted by φ', is a formula in $\mathcal{L}$. The *answer of the query* Q *on* I, denoted by $Q(I)$, is defined as a quantifier-free formula ψ in $\mathcal{L}$ such that $\mathcal{A} \models \varphi' \leftrightarrow \psi$.

If the theory of the structure $\mathcal{A}$ admits quantifier elimination, then each query expression indeed defines a query. The converse is also true. Examples of such theories include the theory of dense-order without endpoints, and the theory of the real closed field. Not all the structures considered in the paper admit quantifier elimination procedures.

Example 3.4 [End72] The theory of $\langle \mathbb{N}, \leqslant, + \rangle$ (a.k.a. Presburger arithmetic) does not admit quantifier elimination. For example, the set of even numbers defined by the formula $\phi(x) \equiv \exists z(x = z + z)$ is not definable by a quantifier-free formula. ■

Though quantifier elimination is a necessary condition for defining the semantics of arbitrary query expressions, Boolean queries are well defined even without quantifier elimination procedure.

Proposition 3.5 For each sentence φ in $\mathcal{L} \cup \sigma$, the answer of the Boolean query $\{() \mid \varphi\}$ on every $(\mathcal{A}, \sigma)$-database is always defined.

We now make our notion of a "query language" more precise. Let $\mathcal{L}$ be a first-order language and $\mathcal{A}$ an $\mathcal{L}$-structure. By *query language* $\mathcal{L}$, we mean the collection of schema and query expression pairs (σ, Q) such that for each $(\mathcal{A}, \sigma)$-database I, $Q(I)$ is always defined. It is not difficult to see that if the theory of $\mathcal{A}$ does not admit quantifier elimination, the corresponding query language $\mathcal{L}$ is possibly non recursive.

First-order queries can also be written in algebraic form. The success of the relational model is largely due to the existence of an algebraic framework, equivalent to the calculus, and which allows efficient optimization and implementation techniques. We recall the algebra for finitely representable relations that was introduced in [GST94]. It plays a fundamental role in the proof techniques for the data complexity on Boolean circuits. We recall the algebraic operators. Suppose R is an n-ary relation represented by a quantifier-free formula, φ, of the form:

$$\varphi \equiv \bigvee_{i=1}^{k} \bigwedge_{j=1}^{\ell} \varphi_{i,j}$$

where the $\varphi_{i,j}$'s are atomic formulas. Then, we can also denote the representation φ as a collection of generalized tuples t_i in the set notation:

$$\left\{ t_i \middle| 1 \leqslant i \leqslant k,\ t_i = \bigwedge_{j=1}^{\ell} \varphi_{i,j} \right\}$$

Furthermore, if I is an instance over schema σ and $R \in \sigma$, we consider the relation $I(R)$ as a set of generalized tuples as above. We also assume that attributes (columns) of relations have names and for each attribute name A, there is a distinct variable x_A associated with it. Attribute names are usually denoted by $A, B, C, \ldots$ (and possibly with subscripts). When the context is clear, we may blur the distinction between variables and attribute names.

Let σ be a schema. The family of *algebraic expressions (over σ)* is defined inductively as usual from the operators: Cartesian product, $\times$, selection[3], σ_F (where F is an atomic formula in $\mathcal{L}$ on attributes), projection, $\pi_{B_1,\ldots,B_k}$ (where $B_1, \ldots, B_k$ are attribute names), set operations, $-, \cap, \cup$, and rename $\rho_{A\to B}$.

We now describe the *semantics* of the algebra. (Note that the operators work directly on generalized tuples, so the semantics is given with respect to generalized tuples.) Suppose that I is an instance of σ, and e is an expression over σ. The *result* of e on I, denoted by $e(I)$, is defined inductively as follows:

(i) – If $e = (R)$, $e(I) = I(R)$ (a set of generalized tuples).
 – If $e = (A : \mathbb{D})$, $e(I) = \{x_A = x_A\}$, where x_A is the variable corresponding to the attribute A and $\mathbb{D}$ is the underlying domain.

(ii) If $e = (e_1 \times e_2)$, then $e(I) = \{t_1 \wedge t_2 \mid t_1 \in e_1(I), t_2 \in e_2(I)\}$.

(iii) If $e = (\sigma_F\ e_1)$, $e(I) = \{t \wedge F \mid t \in e_1(I)\}$, where each attribute name A in F is replaced with the corresponding variable x_A.

(iv) If $e = \pi_{B_1,\ldots,B_k}\ e_1$, then $e(I)$ is obtained from $e_1(I)$ by "eliminating" the variables which do not correspond to attributes B_1 through B_k. One proceeds as follows. Suppose $e_1(I) = \{t_1, \ldots, t_m\}$ and has attributes $A_1, \ldots, A_n$ and $\{C_1, \ldots, C_{n-k}\} = \{A_1, \ldots, A_n\} - \{B_1, \ldots, B_k\}$. We eliminate one by one all existentially quantified variables $x_{C_1}, \cdots, x_{C_{n-k}}$ in each of the formulas $\exists x_{C_1} \cdots \exists x_{C_{n-k}} t_i$. Each tuple t_i then results in t'_i. Finally, $e(I) = \{t'_1, \ldots, t'_m\}$.

(v) – If $e = (e_1 \cup e_2)$, then $e(I) = e_1(I) \cup e_2(I)$.
 – If $e = (e_1 \cap e_2)$, then $e(I) = \{t_1 \wedge t_2 \mid t_1 \in e_1(I), t_2 \in e_2(I)\}$.
 – If $e = (e_1 - e_2)$, then $e(I) = \{t_1 \wedge t_2 \mid t_1 \in e_1(I), t_2 \in (e_2(I))^c\}$, where R^c is the complement of R obtained as follows. Suppose $R = \{t_1, \ldots, t_n\}$ is a set of generalized tuples and for each i, $t_i = \bigwedge_j \varphi_{i,j}$. Then R^c is the formula[4] in DNF which is equivalent to $\bigwedge_i \bigvee_j \neg\varphi_{i,j}$.

[3] The selection σ_F is not to be confused with the schema σ.

[4] Note that the formula may a priori have exponential length in the size of the

(vi) If $e = \rho_{A \to B}\ e_1$, then $e(I) = e_1(I)[x_A/x_B]$ (all occurrences of x_A are replaced by x_B).

We apply the well-known Fourier-Motzkin Elimination method to eliminate one by one all existentially quantified variables for the projection in the case of linear constraints. The Fourier-Motzkin elimination method (see for instance [Sch86], pp. 155–157) works as follows. Consider a generalized tuple t which defines a polyhedron $P(\overline{x}, y) \subseteq \mathbb{Q}^{n+1}$ described by the inequalities (once the coefficients of y have been normalized):

$$\begin{cases} a^\ell \overline{x} + y \leqslant a_0^\ell & \text{for } \ell = 1, ..., L \\ b^k \overline{x} - y \leqslant b_0^k & \text{for } k = 1, ..., K \\ c^i \overline{x} \leqslant c_0^i & \text{for } i = 1, ..., I \end{cases}$$

where $\overline{x} \in \mathbb{Q}^n$, $y \in \mathbb{Q}$. One can show that after the “elimination” of y (i.e. after P has been projected on its first n coordinates), the relation over $\overline{x}$ is exactly:

$$\left\{ \overline{x} \in \mathbb{Q}^n \;\middle|\; b^k \overline{x} - b_0^k \leqslant a_0^\ell - a^\ell \overline{x} \text{ for all } \ell \text{ and } k,\ c^i \overline{x} \leqslant c_0^i \text{ for all } i \right\}.$$

Therefore, $\pi_{\overline{x}}\, t \;=\; \bigwedge\!\!\!\bigwedge_{1 \leqslant k \leqslant L, 1 \leqslant \ell \leqslant L} \quad b^k \overline{x} - b_0^k \leqslant a_0^\ell - a^\ell \overline{x} c^i \overline{x} \leqslant c_0^i.$

It is easy to verify that the algebra, denoted by $\mathrm{ALG}_{\mathcal{L}}$, where $\mathcal{L}$ is some first-order language of the type considered in the paper, is equivalent to first-order logic over the class of structures we consider. The proof (omitted) is similar to that of the equivalence of the classical relational algebra and calculus over finite structures (see [AHV95]).

Theorem 3.6 [GST94] $\mathrm{FO}_{\mathcal{L}} = \mathrm{ALG}_{\mathcal{L}}$.

We illustrate the above result with the following example.

Example 3.7 Consider the following query over a binary relation R with attributes A, B:

$$\left\{ z \mid \exists x \exists y\ (R(x, y) \wedge y = 2x + z) \right\}.$$

The equivalent algebra query is:

$$\pi_{A_2} \sigma_{B = A_1 + A_2} \sigma_{A_1 = A + A} \left(R \times (A_1 : \mathbb{Q}) \times (A_2 : \mathbb{Q}) \right).$$ ■

Optimization techniques for linear queries were investigated in [GL95].

3.3 Data complexity

We next consider the complexity of evaluating queries over finite or finitely representable databases. Various complexity models have been defined in the

original formula $\bigwedge\!\!\!\bigwedge \bigvee\!\!\!\bigvee \neg\varphi_{i,j}$. We prove in the next section that it can be done in polynomial length for the families of databases considered here.

context of arithmetic following two main trends, the arithmetic model and the bit model of complexity [Ren92].

The *arithmetic complexity* [BSS89] refers to a model of computation for real numbers restricted to the arithmetic operations $+, -, \times, \div$ with infinite precision (no rounding errors), with the comparisons $=$ and $<$. These operations correspond to an atomic computation step independent of the size of the numbers involved in the computation. This trend is currently very active [KW95]. This model has been used to develop a descriptive complexity over the reals [GM95] for a special class of models in the spirit of the meta-finite structures of [GG94], associating a finite structure with the real field.

The arithmetic model lacks a sensitivity to the size of numbers, which seems fundamental in the database context. The complexity of queries should be expressed as a function of the size of the data, including of course numbers which can be arbitrarily large. The *bit model* on the other hand reflects the finiteness of the arithmetical computation. It is based on Turing machines with an encoding of all numbers in some finite alphabet on the machine tape. This suppose that all numbers are finitely representable by a string on the tape. We make the following restriction, which is harmless for our purposes.

Proviso: We assume in the sequel, that the numbers stored in the relations or used in the queries, are integers or rationals. In the case of AC^0 results, the numbers are restricted to integers only.

Databases are encoded in a standard way as strings on the machine tape. Rational numbers are encoded as pairs of integers, which are encoded in their binary representation. Let σ be a database schema. An $(\mathcal{A}, \sigma)$-database instance I, is encoded following the *standard encoding* of [GST94]. The *size* of a database is by definition the size of its standard encoding.

The *data complexity* of a query measures the resources (time, space, number of processors, etc.) needed to evaluate the query on a computing device (Turing machine, Boolean circuit, etc.) with respect to the database size. We consider classical complexity classes based on Turing machines such as LOGSPACE and PTIME, classes based on Boolean circuits, with AND, OR, and NEG gates, such as AC^0 with constant depth, a polynomial number of processors and arbitrary fan-in gates, and NC with polylog depth, a polynomial number of processors and bounded fan-in gates (See [Joh90] for a detailed exposition of these classes). We also consider the class $\mathrm{NC}^1 \subset \mathrm{NC}$ based on circuits with logarithmic depth, a polynomial number of processors (bounded fan-in gates).

We define formally the size of a database as follows. This precise definition is fundamental for Boolean circuits.

Definition 3.8 Let R be a relation, and ϕ_R its representation over the language $\{=, \leqslant, +\} \cup \mathbb{Z}$. The formula ϕ_R is *of size* $|\phi_R| \leqslant n$ if ϕ_R contains at most n disjuncts (tuples) and at most n distinct constraints, and the absolute values of the integers occurring in ϕ_R are bounded by 2^n (i.e. the absolute values can be represented in binary notation with n bits).

Note that rational numbers are encoded as pairs of integers. The results about complexity bounds at NC^1 or higher carry over in presence of rationals.

	$\leqslant$	$+$	$\times$	x^y
$\mathbb{N}$	AC^0	AC^0	arithmetical	arithmetical
$\mathbb{Z}$	AC^0	AC^0	arithmetical	arithmetical
$\mathbb{Q}$	AC^0	AC^0	arithmetical	arithmetical
$\mathbb{R}$	AC^0	AC^0	NC	arithmetical

Fig. 1. Data complexity of first-order queries over finite integer databases

We first consider the complexity of queries over finite databases. It is well known in finite model theory that first-order logic with no arithmetic has AC^0 data complexity [AHV95]. The proof is relatively simple and best realized in the context of the relational algebra which is equivalent to first-order queries [Cod70]. We briefly review the proof in the case of the relational algebra over finite structures as it is sketched in [AHV95]. In the case of finite relations, the circuits are constructed uniformly as follows. The gates of the circuit represent pairs of the form $[R, t]$, where R is a relation name (or any algebraic expression, such as $R' \times R''$), and t is a tuple of the same arity as R. The semantics is that the value of a gate $[R, t]$ is 1 iff $R(t)$ holds.

Consider an algebraic query Q. There is a gate of the form $[R, t]$ for each R, either an input relation or a sub-expression of the query Q, and each tuple t which has the proper arity and is built with atomic constants from the input relations. That gives rise to a polynomial number of gates.

The circuit computes the value of $[Q, s]$, for each tuple s of the corresponding arity, starting from the values of the $[R, t]$, where R is an input relation. Most operations are very simple to simulate. For instance, the value of $[R' \times R'', [t', t'']]$ is 1 iff both $[R', t']$ and $[R'', t'']$ have the value 1. The only operation that is slightly more complex is the projection, which requires unbounded fan-in of the OR gates.

The result extends trivially to first-order logic with an order relation, and this holds for all the domains considered in the present paper under the assumption that the numbers occurring explicitly in the databases or in the relations are integers.

Proposition 3.9 First-order queries over finite integer databases in the contexts $\langle \mathbb{N}, \leqslant \rangle, \langle \mathbb{Z}, \leqslant \rangle, \langle \mathbb{Q}, \leqslant \rangle$, or $\langle \mathbb{R}, \leqslant \rangle$ have AC^0 data complexity.

The proof follows easily from the classical finite model theory result. The restriction for integer databases is important. Indeed, the result doesn't hold with rational databases. Consider the following input: $R = \{[\frac{a}{b}, c]\}$, and the query $\forall xy(R(x, y) \rightarrow x = y)$, with the equality predicate. The query is satisfied by R iff $a = b \times c$ holds. This cannot be checked in AC^0 [FSS84]. This example shows that even without order, rational numbers lead to more complex evaluation than AC^0.

The AC^0 bound can be extended to the case of linear queries, as shown in the following proposition.

Proposition 3.10 First-order queries over finite integer databases in the contexts $\langle \mathbb{N}, \leqslant, + \rangle, \langle \mathbb{Z}, \leqslant, + \rangle, \langle \mathbb{Q}, \leqslant, + \rangle$, or $\langle \mathbb{R}, \leqslant, + \rangle$ have AC^0 data complexity.

The proof follows from techniques developed in [GST94] to establish the AC^0 bound for finitely representable linear constraint databases. Details follow in the sequel about finitely representable databases.

For richer languages, there is a serious gap of data complexity. This gap exists already for the complexity of the decision problem of the logical theories, Presburger arithmetic [Pre29,FR79] versus number theory [End72].

Theorem 3.11 First-order queries in the contexts of the natural numbers, $\langle \mathbb{N}, \leqslant, +, \times \rangle$, the integers, $\langle \mathbb{Z}, \leqslant, +, \times \rangle$, and the rationals $\langle \mathbb{Q}, \leqslant, +, \times \rangle$ define mappings in the arithmetical hierarchy.

Proof: It is clear in the context of the natural numbers and the integers, where the language provides exactly the power of the arithmetical hierarchy. Now, for the context of the rational numbers, Robinson [Rob49] proved that the integers are first-order definable over $\mathbb{Q}$. It follows, that in this later context, number theory can be simulated. ■

We analyze in detail queries in the context of $\langle \mathbb{N}, \leqslant, +, \times \rangle$ in Section 4.

On the other hand the complexity is bounded in the real context. This follows from the decidability of the theory of real closed fields [Tar51], and its tractability for a fixed number of variables [BKR86]. The data complexity was shown to be in NC in [KKR90] in a more general context. This result can probably be improved in the present context.

Fig. 1 summarizes the complexity of query languages over various domains and with various arithmetic operations over finite integer databases. The assumption of *integer* databases versus rational databases is necessary only for AC^0 bounds. NC bounds hold in the case of rational databases instead.

We next consider the data complexity over *finitely representable databases* instead of finite databases. It was shown in [KKR90] that first-order for real constraint databases has an NC data complexity upper bound. This tractability result presents a promising future for constraint databases.

	$\leqslant$	$+$	$\times$	x^y
$\mathbb{N}$	AC^0	NC^1	arithmetical	arithmetical
$\mathbb{Z}$	AC^0	NC^1	arithmetical	arithmetical
$\mathbb{Q}$	AC^0	NC^1	arithmetical	arithmetical
$\mathbb{R}$	AC^0	NC^1	NC	arithmetical

Fig. 2. Complexity of first-order queries over finitely representable databases

More recently, Kanellakis and Goldin [KG94] proved that the query language of dense order constraints over the reals (without $+, \times$) has an AC^0 data complexity upper bound when dealing with finitely representable inputs in some normal form. The proof is based on a (finite) relational normal form for finitely representable databases and an equivalent algebra. This technique can be extended to other contexts for $\mathbb{N}, \mathbb{Z}$, and $\mathbb{Q}$. Nevertheless, the result is rather restricted since the normalization is in NC but not in AC^0. Instead of assuming a normalization of all relations, it is possible to consider restricted classes of inputs. In particular, for finitely representable integer databases, the AC^0 bound holds for first-order limited to an order relation as shown in Fig. 2.

An AC^0 bound was obtained in [GST94] for linear constraint queries and databases representable in $\{=, \leqslant, +\} \cup \mathbb{Z}$ with the number of occurrences of $+$ in every constraint uniformly bounded. This result is also used to show the data complexity for finite integer databases (Fig. 1). The result is proved using the algebra for linear constraint databases. The difficult part of the proof is related to the operations of projection and difference, which require some weak form of multiplication.

The initial result of [KKR90] proving an NC bound for polynomial constraints over the reals applies of course to linear constraints over dense orders, such as the rationals or the reals. We next improve the previous bound, and prove that the data complexity of linear queries is actually NC^1. This is the main original complexity result of this paper.

Theorem 3.12 First-order queries over finitely representable databases in the contexts $\langle \mathbb{N}, \leqslant, + \rangle, \langle \mathbb{Z}, \leqslant, + \rangle, \langle \mathbb{Q}, \leqslant, + \rangle$, or $\langle \mathbb{R}, \leqslant, + \rangle$ have NC^1 data complexity.

In the case of constraint databases, the number of tuples (of atomic values) is infinite. Instead of the tuples, the generalized tuples need to be encoded. We next explain how the encoding is done using gates in a circuit.

Without loss of generality, we make a few assumptions to simplify the presentation. Specifically, we assume that Q is a first-order Boolean query whose input consists of a *single binary relation* R. For each natural number n, we exhibit a Boolean circuit C_n, of logarithmic depth (depending only on the query Q) with polynomially many gates in terms of n. The circuit C_n has the property that for each input R with a representation ϕ_R of size smaller than n, the circuit C_n, starting on an encoding $enc(\phi_R)$ of ϕ_R, computes an encoding of $Q(R)$. The proof easily extends to inputs with several relations, of arbitrary arities, and to queries with outputs of arity $\geqslant 1$. The circuits then have many output gates, giving an encoding of (a representation of) the output.

The input (under the previous assumptions) is encoded as follows. We first describe how to encode with $3n^3 + 4n^2$ bits any (quantifier free) formula of $\{=, <, +\} \cup \mathbb{Z}$ with two free variables of size n. (i) Integers are encoded in binary notation with n bits. (ii) Constraints of the form $\alpha x + \beta y \Theta \gamma$, where α, β, and γ are integers whose absolute values are smaller than 2^n, and Θ is $=$ or $<$, are encoded on $3n + 4$ bits as follows:

$$\boxed{\overline{\Theta}\ \overline{\alpha}\ |\alpha|\ \overline{\beta}\ |\beta|\ \overline{\gamma}\ |\gamma|}\ ,$$

where the bit $\overline{\Theta} = 0$ (resp. 1) if Θ is $=$ (resp. $<$); the bit $\overline{\alpha} = 1$ (resp. 0) if α is a positive (resp. negative) integer; $|\alpha|$ is the binary representation of the absolute value of α in n bits; and similarly for β and γ.

Since there are at most n constraints in each tuple and at most n tuples in the binary relation R, the whole encoding of a formula for R of size n requires a sequence of $n \times n \times (3n + 4) = 3n^3 + 4n^2$ bits.

During the computation, the syntactic objects encoded in the circuits can grow in size. For instance, bigger integers may result from adding integers of size n. Similarly, constraints over more than two variables are sometimes needed, as a result of an application of the Cartesian product, for instance. The Cartesian product, along with other operations, also trigger an increase of the number of constraints in each tuple. Therefore, the number of bits allocated to the encoding of integers, constraints, and tuples varies at the different strata (depths) of the circuit. The encoding of bigger integers, constraints over more variables, and tuples containing more constraints, is done in the same manner as above, by adding the required amount of space. Since each first order query can be evaluated using a fixed number of (algebraic) operations, the required additional space can always be figured out once a particular query is given.

Projection and set-difference are the two operations requiring some computations. The others are only based on syntactic manipulation, and result only in simple wiring inside the circuits. Assume for a moment that the operations projection and complement (with which set-difference can be defined), are done with black boxes PROJ and COMP. Then it follows from the detailed technique developed in [GST94], that linear queries can be computed on circuits with arbitrary fan-in, gates AND, OR, NEG, PROJ, and COMP, in constant depth (depending only upon the query), with a polynomial (in the size of the input) number of gates.

In the following, we prove two key lemmas concerning the NC^1 data complexity bound of the two operations projection and complement. Theorem 3.12 then follows from the lemmas 3.13 and 3.14.

The projection operation requires the computation of integer addition, subtraction and multiplication. The addition of two integers can be done in uniform AC^0 with respect to the size of the binary representation of the integers. The multiplication of two integers can be done on bounded fan-in circuits with $O(\log n)$ depth, and $O(n \log n \log\log n)$ size [SS71], and so is in uniform NC^1 with respect to the size of the binary representation of the integers [KR90].

We next prove that the projection can be done in NC^1. More precisely, we prove that for each tuple, there is a circuit of logarithmic depth, with a polynomial number of gates, that computes the projected tuple.

Lemma 3.13 Let S be a set of linear constraints over n variables x_1,...,

x_n, and i a positive integer $\leqslant n$. The projection $\Pi(S)$ of S on variables $\{x_1, ..., x_n\} - \{x_i\}$ is computable in NC^1.

Proof of Lemma 3.13: The NC^1 upper bound for the projection of a tuple relies on the following simple technical claim, which shows how addition and multiplication are used in the computation of the resulting constraints after a set of constraints has been projected onto some components.

Claim: Let S be a set of linear constraints over n variables $x_1, ..., x_n$ of the form: $\sum_{\ell=1}^{n} \alpha_\ell \, x_\ell \, \Theta \, \alpha_0$. Let Π be the projection on variables $\{x_1, ..., x_n\} - \{x_i\}$ for some i. Then the variable coefficients of $\Pi(S)$ are obtained by additions and multiplications of coefficients in $\mathcal{C}(S)$.

The proof of the claim is rather straightforward. Consider the following two constraints in S:

$$\sum_{\ell=1}^{n} \alpha_\ell \, x_\ell \leqslant \alpha_0 \qquad \text{and} \qquad \sum_{\ell=1}^{n} \alpha'_\ell \, x_\ell \geqslant \alpha'_0$$

where $\alpha_i > 0$ and $\alpha'_i > 0$. The resulting constraint using the Fourier-Motzkin method is:

$$\sum_{\ell=1}^{n} (\alpha_\ell \alpha'_i - \alpha_i \alpha'_\ell) \, x_\ell \leqslant (\alpha_0 \alpha'_i - \alpha_i \alpha'_0).$$

Note that in the above constraint the coefficient for x_i is 0 (hence x_i is eliminated). The new constraint verifies the statement of the claim. It is easy to see that for any type of linear constraints the claim holds.

We now see that the projection of S can be done in NC^1. The resulting constraints are obtained by one multiplication and one subtraction. These two operations can be done in NC^1. Moreover, the number of resulting new constraints is at most quadratic in the number of initial constraints, using the Fourier-Motzkin method. ■

The only other operation that requires some care is the set difference. The next lemma is devoted to the complement operation, that can be used to define set difference.

Lemma 3.14 There is a polynomial function $\mathcal{P}$, such that for each relation R of size n, the following conditions hold for the complement, R^c, of R: (i) $|R^c| \leqslant \mathcal{P}(n)$, and (ii) R^c is computable in NC^1 in the size of R.

Proof: Assume that R is a binary relation of size n. The case of a relation of higher arity is dealt with in a similar fashion (dimension 2 is only more intuitive). There are at most n^2 distinct constraints in R (n tuples of n constraints). Since the constraints are linear, there are at most n^4 points, intersection of two lines defined by the constraints in R. Let P be the corresponding set of points. Let P^3 be the set of triples of points in P, such that each triple defines a triangle whose interior do not contain any other point in P. Note that P^3 also contains line segments and points with repetitions in the triples (the triple (p, p, p) denotes a point). P^3 is the set of triangular cells of R. Now let

C^3 be the set of generalized tuples corresponding to the cells in P^3. For each cell, we select those which have an empty intersection with R. This last set defines R^c.

The first claim follows immediately. The number of cells is bounded by a polynomial function in n (see also [Col75] for a more general case). The complement of R, R^c, can easily be computed in NC^1. The set P is computed by the resolution (in parallel) of systems of two two-variate equations. This is performed with a bounded number of sequential multiplications, so in parallel logarithmic time. No computation is done for P^3. C^3 also requires a bounded number of sequential multiplications, and so is done in parallel logarithmic time. The same holds for the last selection step. Therefore, R^c is computed in NC^1 in the size of R. ■

We are now ready to prove Theorem 3.12.

Proof of Theorem 3.12: The proof is by induction on the structure of the formula expressing the query. It follows the steps of the proof of Theorem 5.2 in [GST94], and is therefore omitted. ■

The arithmetical power is obtained exactly in the same cases (polynomial constraints) as before for identical reasons. In the case where more arithmetic is allowed (such as a function x^y), natural numbers become definable over the reals, and so the complexity becomes arithmetical (see Section 4 for details).

Fig. 2 summarizes the data complexity upper bounds. The AC^0 bound holds for finitely representable integer databases. It is still open if the NC result can be improved by showing a Logspace bound.

4 Generic Queries over Finite Databases

In this section, we focus on finite databases and generic queries over finite databases. We study the definability of the PARITY and CONNECTIVITY queries over numeric domains with arithmetic and present a few new results concerning them. Specifically, we show that both queries are definable whenever integers are definable and $+, \times$ are present. Context structures satisfying these two conditions include natural numbers with $+, \times$, integers with $+, \times$, rational numbers with $+, \times$, and real numbers with $+, \times, x^y$. Finally, we prove that for context structures in which integers are definable and $+, \times$ are present, the first order language expresses exactly the set of generic queries (over finite databases) in the arithmetical hierarchy.

We first prove that PARITY is expressible over natural numbers with arithmetic operations. Recall that the PARITY query is defined over a single (finite) relation R and it returns *true* if R has an even cardinality. For simplicity, we assume R is of arity 1. The next theorem shows that addition is inadequate to define parity.

Theorem 4.1 [GST94,PVV95] PARITY is definable in none of the following additive context structures: $\langle \mathbb{N}, \leqslant, + \rangle, \langle \mathbb{Q}, \leqslant, + \rangle$, and $\langle \mathbb{R}, \leqslant, + \rangle$.

Proof: The result was shown in [GST94] in the case of the rational numbers. The proof exhibits an AC^0 data complexity upper-bound for first order logic with order and addition over rational numbers, over inputs that are finite subsets of $\mathbb{N}^k$. Since PARITY is not in AC^0 [FSS84], it follows that PARITY is not definable. Assume now that there is a sentence φ over $\langle \mathbb{R}, \leqslant, + \rangle$ which expresses parity of a set of reals. Then, φ is also a sentence over $\langle \mathbb{Q}, \leqslant, + \rangle$, which also expresses parity of a set of rationals. Therefore parity is not definable in first-order over $\langle \mathbb{R}, \leqslant, + \rangle$. A different technique was exhibited in [PVV95] for the case of real numbers. ∎

Recently, Benedikt, Dong, Libkin, and Wong [BDLW95] have extended the above result and proved that PARITY is not expressible in first order over real numbers with addition and multiplication.

Theorem 4.2 [BDLW95] PARITY is not definable in the context structure $\langle \mathbb{R}, \leqslant, +, \times \rangle$.

The proof, rather complex, relies on powerful model theoretic techniques.

We next prove a positive result, namely that PARITY is definable over the structure of number theory $\langle \mathbb{N}, \leqslant, +, \times \rangle$. This shows that arithmetic has a strong impact on the expressive power of first-order logic for queries that are *generic*, and so independent of the arithmetic.

Theorem 4.3 PARITY is definable in $\langle \mathbb{N}, \leqslant, +, \times \rangle$.

In order to prove Theorem 4.3, we first prove the following weaker result.

Lemma 4.4 PARITY is definable in $\langle \mathbb{N}, \leqslant, +, \times, x^y \rangle$.

Proof: Consider the set $\overline{R} = \{a_1, ..., a_l\}$, and assume without loss of generality that $a_i < a_{i+1}$. The formula $\varphi(R, n)$ defines *the* natural number $n = 2^{a_1} \times 3^{a_2} \times \cdots \times \alpha_l^{a_l}$, where α_l is the l^{th} prime number.

$$\varphi(R, n) \ \equiv \ (\phi(R, n) \wedge \forall n' (\phi(R, n') \Rightarrow n \leqslant n')),$$

where $\phi(R, n)$ is the following formula:

$$div(2^{a_1}, n) \ \wedge \ \forall \alpha \forall \beta \forall a \forall b \begin{pmatrix} \quad prime(\alpha) \\ \wedge \ prime(\beta) \\ \wedge \ succ_{prime}(\alpha, \beta) \\ \wedge \ a \in R \ \wedge \ b \in R \\ \wedge \ succ_R(a, b) \end{pmatrix} (div(\alpha^a, n) \Leftrightarrow div(\beta^b, n)),$$

with the following abbreviations: $div(x, y)$ denotes the relation "x divides y"; $prime(x)$ defines prime numbers; $succ_{prime}(x, y)$ is true if x and y are consecutive primes; $succ_R(x, y)$ is true if x and y are consecutive members of R. All these relations are first-order definable (for first-order definability of relations and functions, see [End72]).

The formula $\varphi'(d,n)$ defines the *biggest* prime divisor d of n. It is easily expressed in first-order.

The formula $\psi(x,y)$ defines the set of pairs of the form (α_k, k), where α_k is the k^{th} prime number. The definition of ψ is given in [End72].

The parity of R is finally expressed by the sentence:

$$\forall n\, \forall d\, \forall k\, ((\varphi(R,n) \wedge \varphi'(d,n) \wedge \psi(d,k)) \Rightarrow div(2,k)). \qquad \blacksquare$$

The proof of Theorem 4.3 now follows from the fact that the exponentiation function is definable in $\langle \mathbb{N}, \leqslant, +, \times \rangle$ [End72].

Interestingly, for the natural numbers, $\mathbb{N}$, $+$ is definable in first order using $\times$ and $\leqslant$, or by divisibility (div) and $\leqslant$ [Rob49]. Therefore, PARITY is definable in the structures $\langle \mathbb{N}, \leqslant, \times \rangle$ and $\langle \mathbb{N}, \leqslant, div \rangle$.

PARITY is also definable with multiplication over the rational numbers.

Theorem 4.5 PARITY is definable in $\langle \mathbb{Q}, \leqslant, +, \times \rangle$.

Proof: This follows from the fact that the set of integers, $\mathbb{Z}$, is definable in the rational field [Rob49]. The theory of the rational field is therefore undecidable, and PARITY can be defined in the same way as over the natural numbers. $\blacksquare$

For the real field, the situation is different since the theory of $\langle \mathbb{R}, \leqslant, +, \times \rangle$ is decidable [Tar51] and PARITY is not definable [BDLW95]. However, PARITY can be defined over the reals with the super-exponentiation function x^y. (But the exponential function e^x is not sufficient [BDLW95].)

Theorem 4.6 PARITY is definable in $\langle \mathbb{R}, \leqslant, +, \times, x^y \rangle$.

Proof: The technique is similar to the previous cases. The set of integers, $\mathbb{Z}$, is definable by the formula: $\varphi(x) \equiv ((-1)^{2x} = 1)$. $\blacksquare$

As shown in the previous proof, the function x^y permits the definition of interesting subsets of the reals such as the integers. Other extensions of the language, for instance with trigonometric functions permit also the definition of the integers. This is the case with the function "cos". The set of integers, $\mathbb{Z}$, is definable in this context by $\varphi(x) \equiv (\cos(2\pi x) = 1)$, and the same technique as before apply to define PARITY. (Note that in the later case the constant π is definable with the function cos.)

The previous definability results extend to the CONNECTIVITY query.

Theorem 4.7 CONNECTIVITY is definable in $\langle \mathbb{N}, \leqslant, +, \times \rangle$.

Proof: Let $G = (V, E)$ be a finite graph. We use a standard encoding of pairs of natural numbers into natural numbers in order to be able to quantify over edges in the graph. Consider the pairing (one-to-one) function from $\mathbb{N} \times \mathbb{N}$ onto $\mathbb{N}$, defined by $J(x,y) = \frac{1}{2}((x+y)^2 + 3x + y)$. Let K and L be the corresponding first and second projections. The functions J, K and L are definable using $\{\leqslant, +, \times\}$ [End72].

	$\leqslant$	$+$	$\times$	x^y
$\mathbb{N}$	no	no	yes	yes
$\mathbb{Z}$	no	no	yes	yes
$\mathbb{Q}$	no	no	yes	yes
$\mathbb{R}$	no	no	no	yes

Fig. 3. Definability of PARITY

The formula $\varphi(E,n)$ defines the smallest integer, n, which is divisible by the k^{th} prime number if $k = J(x,y)$ and there is a *path* from x to y in G.

$$\varphi(E,n) \;\equiv\; (\phi(E,n) \wedge \forall n'(\phi(E,n') \Rightarrow n \leqslant n')),$$

where $\phi(E,n)$ is the formula defined by $\phi(E,n) \;\equiv$

$$\begin{bmatrix} ((k = J(x,y) \wedge G(x,y)) \Rightarrow div(k,n)) \;\bigwedge \\ \forall x\,\forall y \forall z\,\forall k\,\forall k'\,\forall k'' \left(\begin{pmatrix} k \;= J(x,y) \\ \wedge\; k' \;= J(y,z) \\ \wedge\; k'' = J(x,z) \end{pmatrix} \bigwedge \begin{pmatrix} div(k,n) \wedge \\ div(k',n) \end{pmatrix} \right) \Rightarrow div(k'',n) \end{bmatrix}.$$

Connectivity is then expressible by the sentence:

$$\forall x\,\forall y \forall k\,\forall n\,((V(x) \wedge V(y) \wedge k = J(x,y) \wedge \varphi(E,n)) \Rightarrow div(k,n)). \qquad \blacksquare$$

More generally Theorem 3.11 implies the following much stronger result, and in particular, each generic recursive query is expressible in the context of $\langle \mathbb{N}, \leqslant, +, \times \rangle$.

Theorem 4.8 Every generic query in the arithmetical hierarchy is definable in the context structure $\langle \mathbb{N}, \leqslant, +, \times \rangle$.

The proof follows from Theorem 3.11 and the fact that relations of arbitrary arity can be encoded by pairing functions similar to the one used in the proof of Theorem 4.7. Note that Theorem 4.8 subsumes Theorem 4.3 and Theorem 4.7. We presented the later theorems first to illustrate the encoding techniques.

The previous result extends to all contexts where we proved that PARITY was definable.

We have been able to prove that PARITY and CONNECTIVITY are definable in structures whose theories are undecidable, such as the natural numbers with arithmetic. We make the following conjecture.

Conjecture: PARITY is not definable in context structures admitting a decidable theory.

All known results on the PARITY query have been consistent with our conjecture. The same conjecture holds for other queries, such as CONNECTIVITY.

This follows from the reductions presented in Section 5.

5 Queries over Finitely Representable Databases

We now focus on finitely representable databases and queries over them, in particular the queries listed in Section 3. Such queries are concerned with topological properties such as REGION CONNECTIVITY, HOMEOMORPHISM, etc. or arise due to arithmetic such as EUCLIDEAN SPANNING TREE. We prove in this section that none of the queries studied here are expressible in first order in the context structures of $\langle\mathbb{Q},\leqslant\rangle$, $\langle\mathbb{R},\leqslant\rangle$, $\langle\mathbb{R},\leqslant,+,\times\rangle$. The proofs are based on the technique of "first order reductions" [Imm87] from the PARITY query.

We present in the following the notions of "first-order reductions" [Imm87] and then establish reductions among the queries under consideration.

Let $\mathcal{L}'\subseteq\mathcal{L}$ be a first order language and σ a schema. Suppose Q is an n-ary query over a schema $\sigma'=\{R_1,...,R_k\}$ and for each $1\leqslant i\leqslant k$, $\varphi_i(\bar{y}_i,\bar{z})$ is a formula (query) in $\mathcal{L}'\cup\sigma$ with distinct free variables in $\bar{y}_i=y_1,...,y_{\alpha_i}$ and $\bar{z}$ where α_i is the arity of R_i. (The variables $\bar{z}$ serve as parameters.) We use Q as a new operator and let "$Q[\varphi_1,...,\varphi_k]$" denote a relation R_Q of arity $n+m$, where $|\bar{z}|=m$. If I is a database of σ and α is an assignment mapping variables to constants, the first n columns of R_Q holds the answer to the query Q on the input database J defined by: $\forall R_i\in\sigma', J(R_i)=\varphi_i|^{\bar{z}}_{\alpha(\bar{z})}(I)$; and the last m columns have the values $\alpha(\bar{z})$.

We denote the extended language by $(\mathcal{L}'\cup\sigma)+Q$. Intuitively, a formula in $(\mathcal{L}'\cup\sigma)+Q$ can "call" the query Q one or more times. We now present the following definition of a first order reduction.

Definition 5.1 Let Q_1 and Q_2 be two (respectively) n_1-ary and n_2-ary queries over schemas (respectively) σ_1 and σ_2. Let $\mathcal{L}'\subseteq\mathcal{L}$. The query Q_1 is *(first order) $\mathcal{L}'$-reducible* to Q_2 if there exists a formula $\varphi(x_1,...,x_{n_1})$ in $(\mathcal{L}'\cup\sigma_1)+Q_2$ such that for every database instance I of σ_1:

$$\text{for all } a_1,...,a_{n_1}\in\mathbb{R},\quad (a_1,...,a_{n_1})\in Q_1(I) \text{ iff } \mathcal{R}\models\varphi(a_1,...,a_{n_1})\ .$$

Intuitively, Q_1 is $\mathcal{L}'$-reducible to Q_2 if one can systematically construct from an instance I of σ_1 one or more instances of σ_2, apply the query Q_2 one or more times, and obtain the answer to Q_1. Furthermore, all constructions are done in $\mathcal{L}'$.

In the context structure $\mathcal{R}$, $\mathcal{L}=\{\leqslant,+,\times\}$. In particular, the reductions may use the order, addition, and multiplication. Thus, if $\mathcal{S}$ is a subset of $\{\leqslant,+,\times\}$, an $\mathcal{S}$-reduction uses only logical symbols in $\mathcal{S}$ in the reduction in addition to the relation symbols, logical connectives and quantifiers. Especially, we consider $\leqslant$-reductions, $\{\leqslant,+\}$-reductions, and $\{\leqslant,+,\times\}$-reductions. Note that $\leqslant$-reductions are AC^0 reductions. When databases involve only integers, $\{\leqslant, +\}$-reductions are also in AC^0 [GST94]. This is not the case of $\{\leqslant, +, \times\}$-reductions since multiplication is not in AC^0 [FSS84].

Lemma 5.2 Let $\mathcal{L}'' \subseteq \mathcal{L}'$ ($\subseteq \mathcal{L}$) be two first order languages. Then the following hold:

(i) $\mathcal{L}'$-reducibility is transitive; and
(ii) $\mathcal{L}''$-reducibility implies $\mathcal{L}'$-reducibility.

We now consider the first order reductions among queries.

Theorem 5.3 PARITY is first-order $\leqslant$-reducible to MAJORITY; MAJORITY and HALF are first-order $\leqslant$-reducible to each other.

Proof: Let R be a unary input relation for PARITY. clearly, if there is a number α such that the set $S = \{x \in R \mid x \leqslant \alpha\}$ satisfies the following conditions:

$$\text{MAJORITY}(S, R) = \text{MAJORITY}(R - S, R) = \mathit{true}$$

then $\text{PARITY}(R) = \mathit{true}$, i.e., $|R|$ is even.

Now let R_1, R_2 be the input database. To prove the reduction from MAJORITY to HALF, one only needs to find a subset S of R_1 such that $2|S| = |R_2|$, when $|R_2|$ is even. When $|R_2|$ is odd, it is sufficient to find a strict subset S of R_1 such that $2|S| = |R_2 - \{\alpha\}|$, where α is the biggest element in $R_2 - R_1$. Similarly, a subset of R_1 can be defined by a number β such that $S = \{b \in R_1 \mid b \leqslant \beta\}$. Subsets and strict subsets can be easily expressed in first order. For the reduction from HALF, notice that $|R_1| = 2|R_2|$ iff $\text{MAJORITY}(R_1, R_2) = \text{MAJORITY}(R_2 - R_1, R_2) = \mathit{true}$.

It is clear that the above reductions are first order $\leqslant$-reductions. ■

It has already been shown [FSS84,CSV84] that PARITY reduces to (finite graph) CONNECTIVITY. Moreover, several classes of (graph) equivalent problems were presented in [CSV84] under "projection" and "constant-depth truth-table" reductions. The reduction results can also be established in terms of $\leqslant$-reductions. For example, the following shows the $\leqslant$-reduction from MAJORITY to (graph) CONNECTIVITY.

Theorem 5.4 MAJORITY is first-order $\leqslant$-reducible to CONNECTIVITY.

As an interesting aside, Gaifman [Gai81] proved that numerous queries were not definable in first-order logic with only equality using the locality property. This property doesn't carry over in presence of order.

We now consider topological and geometric queries and start with the k-dimensional REGION CONNECTIVITY. Assume that $\sigma = \{R\}$, where R is a k-ary relation symbol. The relation $R \subset \mathbb{R}^k$ defines a *connected region* if every pair of points in R can be linked by a continuous curve lying entirely in R. Note that when $k = 1$, the REGION CONNECTIVITY query reduces to checking if the input unary relation defines a single interval, which is first-order definable.

Theorem 5.5 For each $k \geqslant 2$, MAJORITY is first order $\{\leqslant, +\}$-reducible to the k-dimensional REGION CONNECTIVITY query.

Proof: We consider a 2-dimensional subplane in the k-dimensional space. In the following, we use (i, j) to refer to a point in the plane. Let $R_2 = \{a_1, ..., a_n\}$ and $R_1 = \{b_1, ..., b_m\} \subseteq R_2$ be the inputs for MAJORITY. Assume without loss of generality that

$$0 < a_1 < a_2 < \cdots < a_n \qquad \text{and} \qquad 0 < b_1 < b_2 < \cdots < b_m \ .$$

Intuitively, we construct line segments within a rectangular area bounded by $(0, 0)$ and $(2b_m, a_n)$ as its lower left and upper right corners (resp.), such that (i) the line segments are connected *iff* (ii) the lower left corner is connected to the upper right corner *iff* (iii) $2|R_1| \geqslant |R_2|$, i.e., MAJORITY returns *true*.

Let $a_0 = b_0 = 0$. We construct the following line segments in the plane:

- from $(0, 0)$ to $(2b_m, 0)$;
- from $(0, a_n)$ to $(2b_m, a_n)$; and
- from (b_{i-1}, a_{j-1}) to (b_i, a_j) and from $(b_m + b_{i-1}, a_{j-1})$ to $(b_m + b_i, a_j)$, for $1 \leqslant i \leqslant m$ and $1 \leqslant j \leqslant n$.

Fig. 4(a) shows an example of resulting line segments for $R_1 = \{a_1, ..., a_6\}$ and $R_2 = \{b_1, ..., b_4\} \subseteq R_1$, where $b'_i = b_4 + b_i$ for $1 \leqslant i \leqslant 4$. It is obvious that the set of line segments previously defined is *connected* if and only if

$$\text{MAJORITY}(R_1, R_2) = \textit{true}.$$

Indeed, the line starting from the point $(0, 0)$ reaches the "ceiling" segment only in this last case. ∎

Note that the segments used in the above proof are not definable without $\times$. It suffices, however, to replace each diagonal line by three horizontal and two vertical line segments as shown in Fig. 4(b). It seems that additions are necessary since the input to the MAJORITY query is a pair of arbitrary relations. It is unclear whether an $\leqslant$-reduction exists.

Next we consider AT LEAST and EXACTLY ONE HOLE. Obviously, for one dimensional inputs, both queries are first-order expressible. We prove that for higher dimensions, these queries are analogous to region connectivity.

Theorem 5.6

(i) For each $k \geqslant 2$, the MAJORITY query is first-order $\{\leqslant, +\}$-reducible to k-dimensional AT-LEAST-ONE-HOLE and EXACTLY-ONE-HOLE.

(ii) AT-LEAST-ONE-HOLE is first order $\leqslant$-reducible to REGION CONNECTIVITY.

Proof: For (i), the technique is similar to the previous one. For instance, Fig. 4(c) shows the construction for EXACTLY ONE HOLE for $R_1 = \{a_1, ..., a_6\}$ and $R_2 = \{b_1, ..., b_4\} \subseteq R_1$, where $b'_i = b_4 + b_i$ for $1 \leqslant i \leqslant 4$. For (ii), we note that a relation R has a hole iff its complement is not region connected. ∎

Next we consider the EULERIAN TRAVERSAL query. The input is a set of line segments and the query returns *true* if there is a traversal which goes

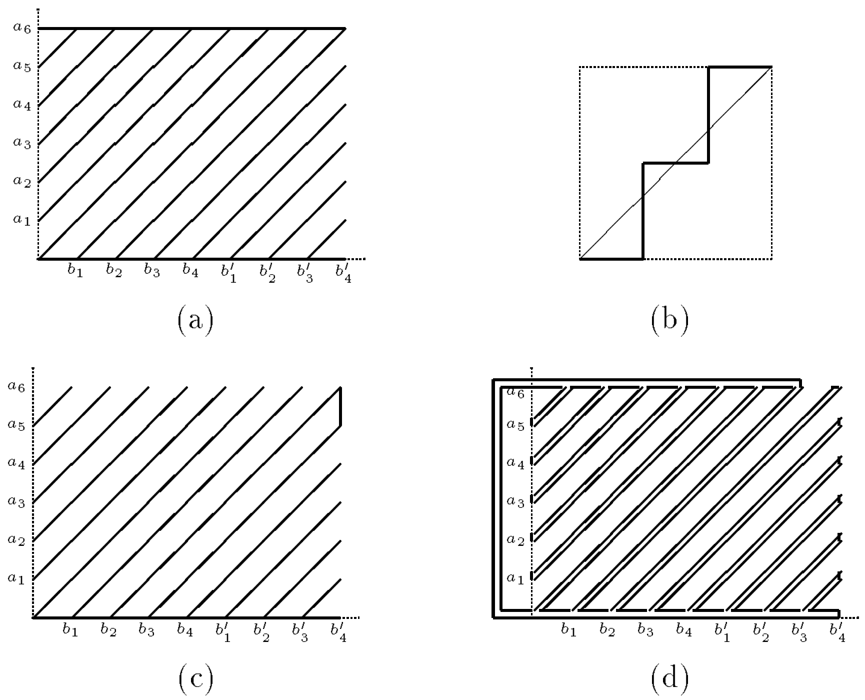

Fig. 4. Illustration of reductions

through each line segment exactly once continuously. In other words, if we view a line as a set of points, a traversal goes continuously through each point exactly once except for a finite set of points (crossings of lines).

Theorem 5.7 HALF is first order $\{\leqslant, +\}$-reducible to EULERIAN TRAVERSAL.

Proof: The proof uses a reduction from the query HALF that is similar to the previous reductions. The basic idea of the reduction is to use pairs of parallel line segments in the reduction, similar to that used in Theorem 5.5. An example of the reduction for $R_1 = \{a_1, ..., a_6\}$ and $R_2 = \{b_1, ..., b_4\} \subseteq R_1$ is shown in Fig. 4(d), where $b'_i = b_4 + b_i$ for $1 \leqslant i \leqslant 4$. Now if HALF returns *true*, the parallel lines originating from $(0, 0)$ go to just below the lowest teeth on the right. Hence a Eulerian traversal exists. Otherwise, if the lines from $(0, 0)$ go too high or too low, the lines are broken into at least two connected parts and Eulerian traversal is impossible. ■

In the next example, we consider another topological property — the HOMEOMORPHISM query. Note that two point sets in the real plane are *homeomorphic* if there is a *bi-continuous* bijection on $\mathbb{R}^2$ which maps one to the other. The *homeomorphism* query is defined over databases with two binary relations. The query returns *true* if the two relations are homeomorphic and *false* otherwise.

Theorem 5.8 HALF is first order $\{\leqslant, +\}$-reducible to HOMEOMORPHISM.

Proof: The proof uses a similar reduction to the one used in the EULERIAN TRAVERSAL query. The primary differences are that (i) here we use closed areas instead of parallel line segments, and (ii) we need to construct the second

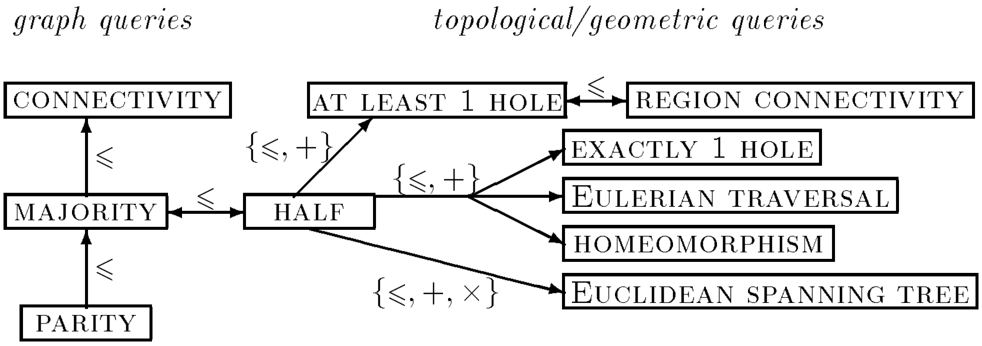

Fig. 5. Summary of Reductions

relation which is a connected and closed area. ■

Finally, we consider EUCLIDEAN SPANNING TREE, whose input is a set of points in $\mathbb{R}^2$ and the output is a set of pairs of points (arity 4) representing the Euclidean spanning tree. We next show that MAJORITY is first order reducible to EUCLIDEAN SPANNING TREE.

Theorem 5.9 HALF is first order $\{\leqslant, +, \times\}$-reducible to EUCLIDEAN SPANNING TREE.

Proof: The idea of the reduction is similar to that in the reduction from MAJORITY to REGION CONNECTIVITY. But instead of creating solid line segments, the reduction produces a "dotted" line segments such that the distance between each pair of consecutive points is tiny with respect to the distance between other points, and also much smaller than the minimal difference among the input numbers for HALF. Hence, a Euclidean spanning tree has to use the dotted line segments. These dotted lines form exactly an Euclidean spanning tree iff HALF is *true*. ■

Fig. 5 summarizes the reduction results. Since PARITY is not definable in $\{\leqslant, +, \times\}$ in the context structure $\langle \mathbb{R}, \leqslant, +, \times \rangle$, it follows that none of the above queries are definable. In particular, it was conjectured in [KKR90] that EUCLIDEAN SPANNING TREE is not expressible in first order. Our results prove that the conjecture holds indeed.

Theorem 5.10 The following queries are not definable in first order for constraint databases: k ($\geqslant 2$) dimensional REGION CONNECTIVITY, AT LEAST ONE HOLE, EXACTLY ONE HOLE; EULERIAN TRAVERSAL, HOMEOMORPHISM, and EUCLIDEAN SPANNING TREE.

6 Conclusion

We have proved that the PARITY and the CONNECTIVITY queries were first-order definable in some contexts, such as in number theory and in some other contexts. These structures are of little interest to constraint databases, since their theories are not decidable, and we believe, that these two queries are not definable in structures admitting a decidable theory.

Constraint databases and query languages can only be defined over structures with decidable theories (moreover with quantifier elimination for non Boolean queries, and a reasonable data complexity). We proved, using first order reduction techniques, that many queries of interest for the purpose of constraint databases, arising in graph theory, computational geometry, or geographical databases are not expressible. In particular, REGION CONNECTIVITY, EXACTLY 1 HOLE, AT LEAST 1 HOLE, EULERIAN TRAVERSAL, HOMEOMORPHISM, and EUCLIDEAN SPANNING TREE are not definable over real polynomial constraint databases.

The results presented in this paper answered many open questions relative to the expressive power of first-order logic with arithmetic. In [KKR90], Kanellakis, Kuper and Revesz conjectured that EUCLIDEAN SPANNING TREE is not definable with polynomial constraints over the reals, that is in $\mathcal{R} = \langle \mathbb{R}, \leqslant, +, \times \rangle$. We proved this is indeed true.

It follows that from a practical point of view, query languages for constraint databases should support aggregate functions, such as counting, and recursion mechanisms. These issues have been addressed recently in the literature. Inflationary Datalog with negation was used in the context of dense order constraint databases [KKR90], and aggregate functions were discussed in [Kup93,CK95].

References

[ACGK94] F. Afrati, S. Cosmadakis, S. Grumbach, and G. Kuper. Expressiveness of linear vs. polynomial constraints in database query languages. In *Proc. Workshop on the Principles and Practice of Constraint Programming*, 1994.

[AHV95] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[BDLW95] M. Benedikt, G. Dong, L. Libkin, and L. Wong. Relational expressive power of constraint query languages. manuscript, 1995.

[BKR86] M. Ben-Or, D. Kozen, and J. Reif. The complexity of elementary algebra and geometry. *Journal of Computer and System Sciences*, 32(2):251–264, April 1986.

[BSS89] L. Blum, M. Shub, and S. Smale. On a theory of computation and complexity over the real numbers. *Bull. Amer. Math. Soc.*, 21:1–46, 1989.

[CH80] A. K. Chandra and D. Harel. Computable queries for relational data bases. *Journal of Computer and System Sciences*, 21(2):156–78, 1980.

[CH82] A. K. Chandra and D. Harel. Structure and complexity of relational queries. *Journal of Computer and System Sciences*, 25(1):99–128, 1982.

[CK73] C. C. Chang and H. J. Keisler. *Model Theory*, volume 73 of *Studies in Logic*. North-Holland, 1973.

[CK95] J. Chomicki and G. Kuper. Measuring infinite relations. In *Proc. 14th ACM Symp. on Principles of Database Systems*, San Jose, May 1995.

[Cod70] E. F. Codd. A relational model of data for large shared data banks. *Communications of ACM*, 13(6):377–387, 1970.

[Col75] G. E. Collins. Quantifier elimination for real closed fields by cylindric decompositions. In *Proc. 2nd GI Conf. Automata Theory and Formal Languages*, volume 35 of *Lecture Notes in Computer Science*, pages 134–83. Springer-Verlag, 1975.

[CSV84] A. K. Chandra, L. Stockmeyer, and U. Vishkin. Constant depth reducibility. *SIAM Journal on Computing*, 13(2):423–439, May 1984.

[Dri82] L. Van den Dries. Remarks on Tarski's problem concerning $(r, +, \times, exp)$. In *Logic Colloquium*, North-Holland, 1982. Elsevier.

[Ehr61] A. Ehrenfeucht. An application of games to the completeness problem for formalized theories. *Fund. Math*, 49, 1961.

[End72] H. B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, Inc., 1972.

[Fag74] R. Fagin. Generalized first-order spectra and polynomial-time recognizable sets. In R. M. Karp, editor, *Complexity of Computation, SIAM-AMS Proceedings, Vol. 7*, pages 43–73, 1974.

[Fag76] R. Fagin. Probabilities on finite models. *Journal of Symbolic Logic*, 41(1):50–58, 1976.

[Fag93] R. Fagin. Finite model theory — a personal perspective. *Theoretical Computer Science*, 116(1):3–31, August 1993.

[FR79] J. Ferrante and C. W. Rackoff. *The Computational Complexity of Logical Theories*, volume 718 of *Lecture Notes in Mathematics*. Springer-Verlag, 1979.

[Fra54] R. Fraïssé. Sur les classifications des systèmes de relations. *Publ. Sci. Univ Alger*, I:1, 1954.

[FSS84] M. Furst, J. B. Saxe, and M. Sipser. Parity, circuits, and the polynomial-time hierarchy. *Math. Syst. Theory*, 17:13–27, 1984.

[Gai81] H. Gaifman. On local and non local properties. In J. Stern, editor, *Proc. Herbrand Symposium Logic Colloquium*, pages 105–135. North Holland, 1981.

[GG94] E. Grädel and Y. Gurevich. Metafinite model theory. In D. Leivant, editor, *Logic and Computational Complexity*, number 960 in Lecture Notes in Computer Science. Springer Verlag, 1994.

[GL95] S. Grumbach and Z. Lacroix. Computing queries on linear constraint databases. In *Proc. Fifth Int. Workshop on Database Programming Languages*, Gubbio, 1995.

[GM95] E. Grädel and K. Meer. Descriptive complexity theory over the real numbers. In *Proc. ACM SIGACT Symp. on the Theory of Computing*, 1995.

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

[GS95] S. Grumbach and J. Su. Dense order constraint databases. In *Proc. 14th ACM Symp. on Principles of Database Systems*, San Jose, May 1995.

[GST94] S. Grumbach, J. Su, and C. Tollu. Linear constraint query languages: Expressive power and complexity. In *Logic and Computational Complexity: International Workshop LCC '94*. Springer-Verlag, 1994.

[Gur88] Y. Gurevich. Logic and the challenge of computer science. In *Current Trends in Theoretical Computer Science*, pages 1–57. Computer Science Press, 1988.

[Hul86] R. Hull. Relative information capacity of simple relational schemata. *SIAM Journal on Computing*, 15(3):856–886, 1986.

[Imm87] N. Immerman. Languages that capture complexity classes. *SIAM Journal on Computing*, 16(4):760–778, 1987.

[Joh90] D. S. Johnson. A catalog of complexity classes. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, chapter 2, pages 67–162. North Holland, 1990.

[KG94] P. C. Kanellakis and D. Q. Goldin. Constraint programming and database query languages. In *Proc. 2nd Conference on Theoretical Aspects of Computer Software (TACS)*, volume 789 of *Lecture notes in computer science*. Springer-Verlag, 1994.

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

[KR90] R. Karp and V. Ramachandran. Parallel algorithms for shared memory machines. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, pages 869–941. North Holland, 1990.

[Kup93] G. M. Kuper. Aggregation in constraint databases. In *Proc. Workshop on the Principles and Practice of Constraint Programming*, pages 176–183, April 1993.

[KW95] Ker-I Ko and K. Weihrauch. Computability and complexity in analysis. In *Workshop at Fern Universitaet*, 1995.

[Pre29] M. Presburger. über die Vollständigkeit eines gewissen systems der arithmetik ganzer zahlen, in welchem die addition als einzige operation hervortritt. In *Comptes rendus du premier Congrès des Mathématiciens des Pays Slaves*, pages 92–101, Warszawa, 1929.

[PVV94] J. Paredaens, J. Van den Bussche, and D. Van Gucht. Towards a theory of spatial database queries. In *Proc. 13th ACM Symp. on Principles of Database Systems*, pages 279–88, 1994.

[PVV95] J. Paredaens, J. Van den Bussche, and D. Van Gucht. First-order queries on finite structures over the reals. In *Proc. IEEE Symposium on Logic In Computer Science*, 1995.

[Ren92] J. Renegar. On the computational complexity and geometry of the first-order theory of the reals. *Journal of Symbolic Computation*, 13:255–352, 1992.

[Rev93] P. Z. Revesz. A closed form for datalog queries with integer (gap)-order constraints. *Theoretical Computer Science*, 116(1):117–149, 1993.

[Rev95] P. Revesz. Datalog queries over set constraint databases. In *Proc. Int. Conf. on Database Theory*, 1995.

[Rob49] J. Robinson. Decidability and decision problems in arithmetic. *Journal of Symbolic Logic*, 14:98–114, 1949.

[Sch86] A. Schrijver. *Theory of Linear and Integer Programming.* Wiley, 1986.

[SS71] A Schönhage and V. Strassen. Schnelle Multiplikation grosser Zahlen. *Comp.*, 7:281–292, 1971.

[Tar51] A. Tarski. *A Decision Method for Elementary Algebra and Geometry.* University of California Press, Berkeley, California, 1951.

[Tra50] B. A. Trakhtenbrot. The impossibilty of an algorithm for the decision problem for finite models. *Doklady Akademii Nauk SSR*, 70:569–572, 1950.

[Ull88] J. D. Ullman. *Database and Knowledge-Base Systems*, volume 1. Computer Science Press, 1988.

[Yao90] F. F. Yao. Computational geometry. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, chapter 7, pages 343–389. North Holland, 1990.