Linear Constraint Databases

(Extended Abstract)

Stéphane Grumbach[1] and Jianwen Su[2] and Christophe Tollu[3]

[1] University of Toronto and INRIA[§]
[2] University of California at Santa Barbara[¶]
[3] Université Paris-Nord, Villetaneuse[‖]

Abstract. We give an AC^0 upper bound on the complexity of first-oder queries over (infinite) databases defined by restricted linear constraints. This result enables us to deduce the non-expressibility of various usual queries, such as the parity of the cardinality of a set or the connectivity of a graph in first-order logic with linear constraints over the reals.

1 Introduction

Since its inception in the early 70's, Codd's relational model of data [Cod70] has been the standard framework of much work on relational databases and query languages. The almost contemporary renewal of "finite model theory" (which dates back to the Ph.D. dissertation of Ron Fagin in 1973) has offered a logical counterpart to this development. So far, Finite Model Theory has been chiefly concerned with the study of extensions of first-order theory and has greatly contributed to a better understanding of the expressibility and the complexity of relational query languages. In short, Finite Model Theory and Codd's relational model have proven to be quite appropriate to the study and design of languages for systems manipulating finite relational data. But, since they compel all relations to be effectively represented, they are no longer adequate to new applications in databases, such as spatial (geographic) or temporal databases, which obviously require the use of infinite sets. Of course, it is unreasonable, from a mere computational point of view, to jump directly from the class of all finite structures to the class of all countable structures. For example, a straightforward extension of the relational model would require infinite representation(s) of infinite data. One must consider more subtle (and more efficient) generalizations, where the data are handled by "finite means".

[§] I.N.R.I.A., Rocquencourt BP 105, 78153 Le Chesnay, France – Stephane.Grumbach-@inria.fr – Work supported in part by Esprit Project BRA AMUSING, and an NSERC fellowship in Canada.

[¶] Dept. of Computer Science, Univ. of California, Santa Barbara, CA 93106, USA – su@cs.ucsb.edu – Work supported in part by NSF grant IRI-9117094 and NASA grant NAGW-3888. A part of work was done while visiting I.N.R.I.A.

[‖] LIPN-URA 1507, Institut Galilée, Université Paris-Nord, 93430 Villetaneuse, France – ct@lipn.univ-paris13.fr

Such generalizations have been the subject of various attempts in recent years; the most promising ones draw their inspiration from already established research areas either in logic (the study of recursive structures in classical model theory and effective algebra) or in computer science (the constraint programming paradigm).

Recursive structures (i.e. relational structures over a countable domain, say the set of nonnegative integers, where every relation is a recursive set of tuples) have been presented by Hirst and Harel [HH93] as a good alternative to finite structures. They have come up with an important trade-off between the class of structures taken as semantics and the class of admissible queries, which poses the challenging problem of exhibiting interesting classes that lie between the recursive and the highly symmetric ones.

The *constraint database model*, introduced by Kanellakis, Kuper and Revesz in their seminal paper [KKR90] and convincingly advocated in [KG94], is another 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 adequate to definite purposes. A generalized (or *finitely representable* in our terminology) relation is a conjunction of such constraints, interpreted in the domain of a given model of the decidable theory. Interesting (and hopefully powerful enough) constraint query languages are therefore obtained by coupling the relational calculus or some version of Datalog with the theory of dense linear orders or the theory of real closed fields. The expressive power and the complexity of first-order logic over finitely representable databases is still far from being clearly understood. That constitutes one of the problems we address in the present paper.

The main results can be summed up as follows ($\mathbb{Z}$ is the set of integers and $\mathbb{Q}$ the set of rationals):

Theorem 5.2 Every first-order query in $\{=, \leq, +\} \cup \mathbb{Q}$ over structures finitely representable in $\{=, \leq, +\} \cup \mathbb{Z}$ with the number of occurrences of $+$ in every constraint uniformly bounded, can be evaluated in AC^0.

Such upper bounds on the complexity of first-order queries have already been produced for dense-order constraints [KG94]. We think that the theorem proven here constitutes a significant improvement since linear constraints are far more expressive than dense-order ones. As a consequence, we get the following corollary.

Theorem 6.1 Parity, graph connectivity, and region connectivity are not first-order definable with linear constraints.

The main theorem (Theorem 5.2) does not carry over in presence of multiplication. We conjecture that its corollary (Theorem 6.1) holds for polynomial constraints, but it is still an open problem.

The paper is structured as follows. In Section 2, we review and discuss some results aiming at initiating an elementary model theory for different classes of countable structures. Section 3 is devoted to basic definitions and examples of finitely representable databases. Section 4 exhibits an algebraic language that is a procedural equivalent of first-order logic over finitely representable databases. The algebra is used in Section 5 to prove the main theorem, from which we infer the non-expressibility results of Section 6. Throughout the paper, we assume familiarity with complexity classes defined by families of boolean circuits, especially NC (functions computable in polylogarithmic time with a polynomial amount of hardware) and AC^0 (functions computable in constant time with a polynomial amount of hardware). For more details on complexity classes, we refer to [Joh90].

2 Restricted Classes of models

In this section, we emphasize some logical consequences of the decision to work with subclasses of countable models. In particular, we investigate conditions under which the compactness theorem or the completeness do not hold. It has been known for long that restricting oneself to finite structures ruins compactness and completeness. On the contrary, extending the semantics to all countable structures ensures compactness (a direct consequence of the Löwenheim-Skolem Theorem). In this section, we fix a purely relational signature $\sigma = \{R_1, \ldots, R_n\}$ (sometimes, one needs that at least one of the R_i's is of arity ≥ 2). All structures will be of the form $\mathcal{A} = \langle A, R_1, \ldots, R_n \rangle$, with A some countable set (say a subset of natural numbers). If A is finite, one recovers the usual notion of a *finite* structure. If A and all R_i's are recursively enumerable (respectively recursive, primitive recursive), then $\mathcal{A}$ is said to be *recursively enumerable* (respectively *recursive*, *primitive recursive*). Let Str_{fin} (respectively $Str_{r.e.}$, Str_{rec}, $Str_{p.r.}$) denote the set of all finite (respectively recursively enumerable, recursive, primitive recursive) structures, and V_{fin} (respectively $V_{r.e.}$, V_{rec}, $V_{p.r.}$) denote the set of all σ-sentences true in all structures of Str_{fin} (respectively $Str_{r.e.}$, Str_{rec}, $Str_{p.r.}$). The following theorem, due to Mostowski [Mos57] and Vaught [Vau60], establishes that, for any reasonable class of "constructive structures", the completeness theorem fails:

Theorem 2.1 (Mostowski [Mos57] and Vaught [Vau60]) Let V be a set of σ-sentences. If $V_{r.e.} \subseteq V \subseteq V_{fin}$, then V is not recursively enumerable. Moreover, if $V_{r.e.} \subseteq V \subseteq V_{p.r.}$, then V is not arithmetical.

Let us now consider with more details the class Str_{rec}. For $\sigma = \{E\}$, with E binary, it has already been the focus of some attention in the past, particularly from combinatorists. Indeed, it has been proved that switching from finite graphs to recursive ones can tremendously increase the data complexity of usual problems. For instance, the existence of a Euler path (which can be decided in polynomial time in the finite case) becomes Π^0_3-complete, thus undecidable [Bea76], while Hamiltonicity (a well-known NP-complete problem for finite graphs) becomes Σ^1_1-complete, thus even not in the arithmetical hierarchy [Har91].

More recently, Hirst and Harel [HH93] studied the recursive structures from a database point of view. Some of their results are worth mentioning. It is known that very primitive relational operators, e.g. projections, do not preserve the recursiveness of relations: if $T(x, y, z) \subset \omega^3$ is the primitive recursive relation expressing that "the y^{th} Turing machine halts on input z in x steps", then $\exists x\, T(x, y, z) \subset \omega^2$ expresses the halting problem. As a consequence, if one wants queries to be computable, even the relational calculus (i.e. first-order logic) is too expressive a language. Hirst and Harel show that, over the class of all recursive countable databases, quantifier-free first-order logic is complete with respect to the class of computable and generic (a consistency criterion expressing commutation with isomorphisms) queries. Consequently, they define a drastically restricted subclass of recursive databases, called "highly symmetric", whose behavior with respect to completeness (a version of Chandra and Harel's QL [CH80]) and BP-completeness [Ban78, Par78] (first-order logic) resemble the class of finite databases. Thus, they have come up with an important trade-off between the class of structures taken as semantics and the class of admissible queries, which poses the challenging problem of exhibiting interesting classes that lie between the recursive and the highly symmetric ones. Seemingly, the constraint database model offers a framework for the definition of such classes.

In their notes on recursive model theory [HH94], Hirst and Harel prove that the compactness theorem fails for the class of all countable recursive structures. Their argument does not lend itself naturally to arbitrary subclasses of countable structures. J. Väänänen [Va94] suggested that the compactness theorem should fail for any subclass of countable structures containing all finite structures and no infinite countable structure elementary equivalent to a fixed (infinite) locally finite structure (a structure is locally finite if every sentence of its theory has a finite model).

3 Linear Constraint Databases

Constraint databases may be defined over various sorts of constraints, such as dense-order constraints, polynomial constraints over the reals, etc. We next introduce a general paradigm independent of the choice of the constraints. Let $\mathcal{L}$ be a first-order language with equality and D some non empty set. We consider an $\mathcal{L}$-structure, $\mathcal{D}$, with universe D. $\mathcal{D}$ is called the *domain-structure*. Finally, let $\mathcal{T}$ be the first-order theory of $\mathcal{D}$.

Consider for instance, $\mathcal{L} = \{\leq, +\} \cup \mathbb{Q}$. The structure we shall be concerned with in the present paper is $\mathcal{D} = \langle \mathbb{Q}, \leq, +, (q)_{q \in \mathbb{Q}} \rangle$, the structure of the linearly ordered set of the rational numbers with addition and all rational constants, and $\mathcal{T}$ is the theory of dense orders without endpoints and with addition. (Another traditional example is $\mathcal{L} = \{\leq, +, \times, 0, 1\}$, $\mathcal{D} = \langle \mathbb{R}, \leq, +, \times, 0, 1 \rangle$ (the field of reals) and $\mathcal{T}$ is the theory of the ordered real closed fields.)

Let $\sigma = \{R_1, \ldots, R_n\}$ be a signature (or a database schema) such that $\mathcal{L} \cap \sigma = \emptyset$, where $R_1, \ldots, R_n$ are relation symbols. We distinguish between *logical predicates* (e.g., $=, \leq$) in $\mathcal{L}$ and *relations* in σ. We next introduce a restricted definition of *finitely representable structures* [GS94]. We consider expansions of $\mathcal{D}$ to σ. Intuitively, the relations in σ constitute a database *in the context* of $\mathcal{D}$.

Definition 3.1 Let $S \subseteq D^k$ be some k-ary relation. The relation S is *finitely representable in $\mathcal{L}$ over $\mathcal{D}$* (*$\mathcal{L}$-representable* for short) if there exists a quantifier free formula $\varphi(x_1, \ldots, x_k)$ in $\mathcal{L}$ with k distinct free variables $x_1, \ldots, x_k$ such that:

$$\mathcal{D} \models \forall x_1 \cdots x_k (S(x_1, \ldots, x_k) \leftrightarrow \varphi(x_1, \ldots, x_k)).$$

Let $\mathcal{A}$ be an expansion of $\mathcal{D}$ to σ. The structure $\mathcal{A}$ is finitely representable (over $\mathcal{D}$) if for every relation symbol R in σ, $R^{\mathcal{A}}$ is $\mathcal{L}$-representable (over $\mathcal{D}$).

Kanellakis, Kuper, and Revesz [KKR90] introduced the concept of a 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 the remainder of the paper, we focus on the language $\mathcal{L} = \{\leq, +\} \cup \mathbb{Q}$ and the $\mathcal{L}$-structure $\mathcal{D} = \langle \mathbb{Q}, \leq, +, (q)_{q \in \mathbb{Q}} \rangle$. Therefore, constraints will be composed of linear equations or inequalities of the form:

$$\sum_{i=1}^{p} a_i\, x_i = a_0, \qquad \sum_{i=1}^{p} a_i\, x_i \leq a_0$$

where the x_i's denote variables and the a_i's are integer constants (note that rational constants can always be avoided in linear equations and inequalities).

A *(database) instance (of σ)* is a mapping which associates with each k-ary relation symbol R in σ a quantifier-free formula in disjunctive normal form with k distinct variables. Clearly, each instance of σ corresponds to the restriction of a finitely representable structure to σ. In practice, we assume that the databases contain the formula defining their relations. Instances will be denoted by I, J, etc.

Note that the class $\mathcal{K}$ of σ-instances is effectively enumerable if the cardinality of the language $\mathcal{L}$ is countable. Moreover, if $\mathcal{D}$ is recursive, then instances are recursive. $\mathcal{K}$ has interesting closure properties. It is closed under finite union and intersection and moreover under complementation. This differs from finite model theory (the complement of a finite model is not finite). Our main goal is to investigate the expressive power of first-order logic over the class of linear constraint databases. We consider partial recursive classes of $\mathcal{L}$-representable databases and ask whether they can be captured by a first-order sentence in $\mathcal{L}$.

In the main theorem, we restrict our attention to a family of database instances, called "k-bounded" instances. Intuitively, k-bounded linear instances are defined with equations and inequalities with bounded variable factors. We shall prove that first order queries over k-bounded instances can be evaluated in AC^0 in terms of the database size (data complexity). Following is a formal definition of k-boundedness.

Definition 3.2 Let $k \geq 0$ be an integer. An atomic formula is *k-bounded* if it is in $\{\leq, +\} \cup \mathbb{Z}$ (no rationals) and contains at most k occurrences of the (function) symbol "+". A quantifier-free formula is *k-bounded* if each atomic formula in it is k-bounded. Finally, an instance of signature σ is *k-bounded* if for each relation symbol R, the associated quantifier-free formula is k-bounded. We denote by $\mathcal{K}_k(\sigma)$, or simply $\mathcal{K}_k$, the family of all k-bounded instances over σ.

A k-bounded constraint has the following form:

$$\sum_{i=1}^{p} a_i \, x_i \Theta a_0$$

where Θ is a predicate, the a_i's are integers, and $\sum_{i=1}^{p} |a_i| \leq k+1$ (where $|a_i|$ denotes the absolute value of a_i).

Note that when $k = 0$, $\mathcal{K}_0$ is exactly the set of dense order constraints which were studied in [KKR90, KG94, GS94]. For this class of constraints, an upper bound on the complexity of the first-order queries expressed in the language $\{\leq\} \cup \mathbb{Q}$ is known:

Theorem 3.1 [KG94] The data complexity of first order logic in the language $\{\leq\} \cup \mathbb{Q}$ over the family $\mathcal{K}_0$ of dense order instances is in AC^0.

The proof of this result is based on an encoding of dense order instances into finite instances. This is possible since dense order instances admit very simple geometrical decompositions in terms of atomic "cells" [Col75] of simple shapes. Note that the encoding itself is not in AC^0. A specific algebra working on finite structures is introduced in [KG94], which simulates the manipulation of dense order instances.

The geometrical decomposition into atomic cells of basic shapes is also possible in the case of linear instances as shown in [GS94]. The main theorem of this paper doesn't require any encoding into finite structures, but it assumes also a normal form for the input instances based on a geometrical decomposition into atomic cells. Consider binary input relations (sets of subsets of $\mathbb{Q}^2$). They can be decomposed into "triangles", i.e. sets of tuples, where each tuple consists of at most *three* constraints. As shown in [GS95], this can be generalized to arbitrary arity: relations of arity k, representing regions in dimension k, can be represented by using only tuples with at most $k+1$ constraints. Moreover, the size of the normal form of a relation is polynomially bounded by the size of the initial relation [GS95].

The normal form is fundamental to maintain the size of the boolean circuits computing the queries reasonable. The normal form of the input is not maintained during the query evaluation, but thanks to the k-boundedness of the input, we shall be able to fix a bound on the number of constraints in each tuple at each step of the computation (see Section 5).

4 First-order Query Languages

We define $\mathrm{FO}_{\mathcal{L}}$ as first-order logic with linear constraints, i.e. over the language $\mathcal{L} = \{\leq, +\} \cup \mathbb{Q}$. We introduce in this section an algebra $\mathrm{ALG}_{\mathcal{L}}$ for finitely representable databases, and prove its equivalence with $\mathrm{FO}_{\mathcal{L}}$. This algebra is similar to Codd's algebra for finite relations [Cod70], but the operators apply to finite representations of possibly infinite sets. The algebra consists of the following operations: cartesian product, $\times$, selections ($\sigma_=$, $\sigma_<$, and σ_+), projection, π, set operations (union, $\cup$, intersection, $\cap$, and set difference, $-$), and rename, ρ.

The algebra operations are performed on sets of generalized tuples, i.e. on quantifier-free formulas in *disjunctive normal form*. But unlike Kanellakis and Goldin [KG94], we do not assume special encoding for relations and generalized tuples. On the other hand, our algebra can also be viewed as a simplified sublanguage of the algebra of Paredaens, Van den Bussche and Van Gucht [PBG94] (which also includes multiplication).

We now define the algebra 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 also denote the representation φ as a collection of generalized tuples t_i in the set notation:

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

Furthermore, if I is an instance over signature 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.

Definition 4.1 Let σ be a signature. The family of *algebraic expressions (over σ)* is defined inductively as follows:

1. (R) and $(A : \mathbb{Q})$ are *atomic* expressions, where $R \in \sigma$ is a relation symbol, and A is an attribute name. The set of attributes is the set of attributes of R or $\{A\}$, respectively.

Suppose now that e_1 and e_2 are two algebraic expressions.

2. (Cartesian product) If e_1 and e_2 have disjoint sets of attribute names, then $(e_1 \times e_2)$ is also an expression.
3. (Selection) If F is a selection formula (defined below) involving only attribute names of e_1, then $(\sigma_F\ e_1)$ is an expression. A *selection formula* is an atomic formula of one of the following three forms:

$$t_1 = t_2, \qquad t_1 \leq t_2, \qquad t_1 + t_2 = t_3,$$

where t_1, t_2, t_3 are attribute names or constants (in $\mathbb{Q}$).

4. (Projection) If e_1 has attributes $\{A_1, \ldots, A_n\}$, and moreover $\{B_1, \ldots, B_k\} \subseteq \{A_1, \ldots, A_n\}$, then $(\pi_{B_1,\ldots,B_k}\, e_1)$ is an expression.
5. (Set operations) If e_1, e_2 have exactly the same set of attributes, then $(e_1 - e_2)$, $(e_1 \cap e_2)$, and $(e_1 \cup e_2)$ are expressions.
6. (Rename) If A, B are two attribute names and A is an attribute of e_1, then $(\rho_{A \to B}\ e_1)$ is also an expression.

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:

1. (a) If $e = (R)$, $e(I) = I(R)$ (a set of generalized tuples).
   (b) If $e = (A : \mathbb{Q})$, $e(I) = \{x_A = x_A\}$, where x_A is the variable corresponding to the attribute A.
2. If $e = (e_1 \times e_2)$, then $e(I) = \{t_1 \wedge t_2 \mid t_1 \in e_1(I), t_2 \in e_2(I)\}$.
3. 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.
4. 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 apply the well-known Fourier-Motzkin Elimination method [Sch86] (see below) to 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, $e(I) = \{t'_1, \ldots, t'_m\}$.
5. (a) If $e = (e_1 \cup e_2)$, then $e(I) = e_1(I) \cup e_2(I)$.
   (b) If $e = (e_1 \cap e_2)$, then $e(I) = \{t_1 \wedge t_2 \mid t_1 \in e_1(I), t_2 \in e_2(I)\}$.
   (c) 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[9] in disjunctive normal form which is equivalent to $\bigwedge_i \bigvee_j \neg\varphi_{i,j}$.
6. 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).

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):

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

[9] Note that the formula may a priori have exponential length in the size of the original formula $\bigwedge \bigvee \neg\varphi_{i,j}$. We prove in the next section that it can be done in polynomial length for the families of databases considered here.

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:

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

Therefore:

$$\pi_{\overline{x}}\, t = \bigwedge_{1\leq k\leq L, 1\leq \ell\leq L} b^k\overline{x} - b_0^k \leq a_0^\ell - a^\ell\overline{x} \;\wedge\; \bigwedge_{1\leq i\leq I} c^i\overline{x} \leq c_0^i.$$

It is easy to verify that the algebra, denoted by $\mathrm{ALG}_{\mathcal{L}}$, is equivalent to first-order logic over the class of structures we consider. The proof (which is omitted) is quite similar to that of the equivalence of the classical relational algebra and calculus over finite structures (see [AHV94]).

Theorem 4.1 $\mathrm{FO}_{\mathcal{L}} = \mathrm{ALG}_{\mathcal{L}}$.

We illustrate the above result with the following example.

Example 4.1 Consider the following query over a binary relation R with attributes A, B:

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

The equivalent algebra query is:

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

Remark. The combination of selections and cartesian products can yield complicated form of selections. For instance, $\sigma_{B=kA+c}(e)$ (e is an expression with attributes A, B) can be expressed as:

$$\pi_{A,B}\ \sigma_{B=A_{k-1}+c} \sigma_{A_{k-1}=A_{k-2}+A} \cdots \sigma_{A_1=A+A} \left(e \times (A_1 : \mathbb{Q}) \times \cdots \times (A_{k-1} : \mathbb{Q}) \right).$$

Finally, we discuss the complement operation used in defining the semantics for the set difference operation. Computing the complement of a relation R is generally a costly operation. The naive approach to converting a formula $\bigwedge \bigvee \neg\varphi_{i,j}$ of size n into disjunctive normal form might generate a formula whose length is exponential in n. However, there are special cases where efficient algorithms exist.

Example 4.2 Suppose R is a set of binary generalized tuples consisting of linear constraints with a fixed set of k distinct (rational) slopes $\alpha_1, \ldots, \alpha_k$. We can view the constraints in R as dividing $\mathbb{Q}^2$ into many "cells" and R as a collection of these cells. Since each cell is a convex polygon, every cell can be defined using at most $2k$ constraints. It can then be verified that there exists a representation of the complement of R, where each tuple involves at most $2k$ constraints. Let S be the set of all possible constraints (involving n variables) in R or obtained from constraints in R by changing the logical predicates and define:

$$U = \left\{ t_1 \wedge \cdots \wedge t_{2k} \;\middle|\; \text{for each } 1 \leq i \leq 2k,\ t_i \in S \right\}.$$

Then, the complement of R can be defined as:

$$R^c = \left\{ t \in U \mid \forall \overline{x}\ t(\overline{x}) \Rightarrow \neg R(\overline{x}) \right\}.$$

Therefore, the complement can be computed in polynomial time for each fixed k. $\square$

5 Complexity

In this section, we analyze the data complexity of first-order queries. We present two results: (i) a known NC bound [KKR90] in the general case, and (ii) a new AC^0 bound for a restricted class of inputs, namely k-bounded instances for a fixed k. The proof of this last result relies on the algebra introduced in the previous section.

It was shown in [KKR90] that first-order queries with polynomial constraints (over the real numbers) have NC data complexity. This result follows from techniques, first introduced in [BKR86], showing that the theory of real closed fields of fixed dimension (number of variables) can be decided in NC. The same upper bound of course holds in the case of linear constraints.

Theorem 5.1 [BKR86, KKR90] $\mathrm{FO}_{\mathcal{L}}$ is in NC over the class of linear constraint inputs.

We next present the main theorem of this section. It applies to a restricted class of inputs that is of practical interest. Recall that a *k-bounded linear constraint input* is a relation that is finitely representable by a quantifier free formula in DNF, such that in each atomic formula occurring in it there are at most k occurrences of the addition symbol, and all constants are integers.

The complexity upper bound is defined with respect to the *size* of the representation of the input instances in *normal form*. Note for instance that each binary relation in normal form consists of tuples, each of which has at most 3 constraints. The *size* of an input relation is defined as follows. Let R be a relation, and ϕ_R its representation in normal form over the language $\{=, \leq, +\} \cup \mathbb{Z}$. The formula ϕ_R is of size $|\phi_R| \leq n$ if ϕ_R contains at most n disjuncts (tuples), and the absolute values of the integers occurring in ϕ_R are bounded by 2^n (i.e. the absolute values can be represented with n bits).

We can now state the theorem that applies to k-bounded linear constraint inputs in normal form.

Theorem 5.2 For each (fixed) integer $k \geq 0$, $\mathrm{FO}_{\mathcal{L}}$ is in uniform AC^0 over the class of k-bounded linear constraint inputs.

This result extends the now classical theorem showing that the relational algebra has AC^0 data complexity over finite structures. Before presenting the proof of Theorem 5.2, let us briefly review the proof in the case of the relational algebra over finite structures as it is sketched in [AHV94]. 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 $[R, t]$ 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 an algebraic expression that is 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.

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. 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 constant depth (depending only upon the query Q and the degree k of boundedness of the inputs) with polynomially many gates in terms of n. The circuit C_n has the property that for each k-bounded input R with a representation ϕ_R in normal form of size smaller than n, the circuit C_n, starting on an encoding $enc(\phi_R)$ of ϕ_R, computes the value of $Q(R)$.

The proof easily extends to inputs with several relations, of arbitrary arities, and to queries with outputs of arity ≥ 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 $9n^2 + 12n$ bits any (quantifier free) formula of $\{=, <, +\} \cup \mathbb{Z}$ with two free variables, in normal form and 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 the relation is binary and in normal form, disjuncts (tuples) are sequences of at most three constraints. Moreover, there are at most n disjuncts. Therefore, the whole encoding of a formula of size n requires a sequence of $n \times 3 \times (3n + 4) = 9n^2 + 12n$ 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.

The projection operation requires the computation of addition, and repeated addition (bounded multiplication). We first prove that (i) the addition of two integers, and thus (ii) the multiplication of an integer by a given constant can be done in uniform AC^0 with respect to the size of the binary representation of the integers.

Lemma 5.3 The addition of two binary integers of size $\leq n$, $a_n a_{n-1} \cdots a_1 a_0$, and $b_n b_{n-1} \cdots b_1 b_0$, can be done by constant-depth circuits with $n+1$ output gates and at most $\mathcal{P}(n)$ gates, where $\mathcal{P}$ is a polynomial.

Proof: Assume that:

$$a_n a_{n-1} \cdots a_1 a_0 + b_n b_{n-1} \cdots b_1 b_0 = c_{n+1} c_n \cdots c_1 c_0$$

The boolean circuit is constructed uniformly with the following formulas:

$$c_0 = \neg(a_0 \Leftrightarrow b_0)$$

$$c_k = (a_k \Leftrightarrow b_k) \Leftrightarrow \bigvee_{i=0}^{i=k-1} \big((\bigwedge_{j=i+1}^{j=k-1} (a_j \vee b_j)) \wedge (a_i \wedge b_i) \big)$$

for each $1 \leq k \leq n$, and

$$c_{n+1} = \bigvee_{i=0}^{i=n} \big((\bigwedge_{j=i+1}^{j=n} (a_j \vee b_j)) \wedge (a_i \wedge b_i) \big).$$

where "$\Leftrightarrow$" is an abbreviation for a circuit of depth 3 using only $\neg$, $\wedge$, and $\vee$ nodes: $x \Leftrightarrow y \equiv (x \wedge y) \vee (\neg x \wedge \neg y)$. The depth of the circuit is 7, and the number of nodes is $O(n^3)$. $\square$

Remark. On the other hand, addition of rational numbers (coded as pairs of natural numbers) is not in AC^0. Indeed, this would imply that multiplication of natural numbers is: just consider $\frac{1}{x} + \frac{1}{y} = \frac{y+x}{y \times x}$. As a consequence, our proof does not carry over to the case where databases are defined with rational numbers as parameters. This follows from the fact that addition of parameters coming from the input constraints is required to compute an application of projection.

We next prove that the projection can be done in AC^0. More precisely, we prove that for each tuple, there is a circuit of fixed depth, with a polynomial number of gates, that computes the projected tuple.

Lemma 5.4 Let S be a k-bounded set of linear constraints over n variables $x_1, x_2, ..., x_n$ for some k, and i a positive integer $\leq n$. The projection $\Pi(S)$ of S on variables $\{x_1, ..., x_n\} - \{x_i\}$ is computable in AC^0.

In Lemma 5.4, the set S denotes a single k-bounded tuple, i.e. the total number of occurrences of the addition symbol in each constraint in S is bounded by k. It follows easily that the projection of an entire k-bounded relation can be done in AC^0. The circuit contains essentially copies of the circuit that computes the projection individually for each tuple.

Proof of Lemma 5.4: The AC^0 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 k-bounded set of linear constraints over n variables $x_1, ..., x_n$ of the form: $\sum_{\ell=1}^{n} \alpha_\ell \, x_\ell \, \Theta \, \alpha_0$, and let $\mathcal{C}(S)$ be the set of variable coefficients (namely, α_ℓ for each $\ell \geq 1$), and $\mathcal{C}_0(S)$ the set of constant coefficients (namely, α_0), in the constraints of S. Let Π be the projection on variables $\{x_1, ..., x_n\} - \{x_i\}$ for some i. Then the following holds:

- The variable coefficients of $\Pi(S)$ are obtained by additions and multiplications of variable coefficients in $\mathcal{C}(S)$, and
- The constant coefficients of $\Pi(S)$ are obtained by multiplications of a constant coefficient in $\mathcal{C}_0(S)$ with a variable coefficient in $\mathcal{C}(S)$, and additions.

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

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

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

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

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 AC^0. Since S is a k-bounded set of linear constraints, the variable coefficients are not larger than the constant k. Therefore, the resulting constraints are obtained by multiplication with a constant (integer) not larger than k, and by addition. These two operations can be done in AC^0 (it follows from Lemma 5.3). 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 5.5 Let k be a (fixed) positive integer and $\mathcal{K}_k$ be the class of k-bounded linear constraint relations. There is a polynomial function $\mathcal{P}$, such that for each relation R in $\mathcal{K}_k$ of size n, the following conditions hold for the complement, R^c, of R: (i) $|R^c| \leq \mathcal{P}(n)$, and (ii) R^c is computable in AC^0 in the size of R.

Proof: (sketch) We only give the proof for a binary relation R. It will be clear how to extend it to an arbitrary arity. Since R is a k-bounded linear constraint relation, it follows that the number of different slopes of line segments in R is bounded by some $k' \leq k^2$. Therefore the "cells" (in the sense of [Col75]) can be defined by tuples containing not more than $2k'$ constraints. Assume that R is defined with ℓ different constraints. It can be easily verified that the constraints needed to define the cells for the complement R^c are the existing constraints, and their variants obtained by replacing the predicate in each constraint, by $=$, $<$, or $>$. This generates at most 3ℓ constraints. Since no other constraint is required, every cell in the plane is therefore definable with at most $2k'$ constraints. There are 3ℓ possibilities for each constraint, thus it leads to at most $(3\ell)^{2k'}$ possible cells. The number of cells is therefore polynomial (see also [Col75]), and the complement can easily be computed in AC^0 (using only operations in $\mathrm{ALG}_{\mathcal{L}}$ as shown in Example 4.2). □

We are now ready to prove Theorem 5.2.

Proof of Theorem 5.2: The proof is by induction on the structure of the formula expressing the query. We can always assume that the boolean query is of the form $\pi_\emptyset e$, where e is some algebraic expression (i.e. a test of emptyness).
Basis: Assume $e = R$. To verify that $\pi_\emptyset R$ is false, it suffices to check that each tuple in R defines an empty set. This can be done by applying the Fourier-Motzkin method. It follows from Lemma 5.4, that this can be done in AC^0.

In the sequel, we prove by induction that we can compute in AC^0, an encoding of $e(R)$ starting from an encoding of R for each subexpression e.
Induction: The induction step depends on the last algebraic operations performed. We first consider the gates of the circuit, and then illustrate how it is wired. We next establish the upper bounds of (i) the number of constraints in each tuple, and (ii) the number of tuples in the new relations, resulting from the application of algebraic operations. Let e_i be a k-bounded relation of n_i tuples, each tuple consisting of k_i constraints ($i = 1, 2$). Note that both the number n_i

of tuples and the number k_i of constraints may not be exact. However they are upper bounds. Physically, the circuits contain the space to encode n_i tuples of k_i constraints.

1. If $e = (e_1 \times e_2)$, then e is a k-bounded relation containing $n_1 \times n_2$ tuples, each of which is represented with $k_1 + k_2$ constraints.
2. If $e = (\sigma_F\ e_1)$, then e is a k-bounded relation containing n_1 tuples, each of which is represented with $k_1 + 1$ constraints.
3. If $e = \pi\ e_1$, where π just eliminates a single variable, then e is a k^2-bounded relation containing n_1 tuples, each of which is represented with at most $(k_1)^2$ constraints.
4. If $e = (e_1 \cup e_2)$, then e is a k-bounded relation containing $n_1 + n_2$ tuples, each of which is represented with $max(k_1, k_2)$ constraints.
5. If $e = (e_1 \cap e_2)$, then e is a k-bounded relation containing $n_1 \times n_2$ tuples, each of which is represented with $k_1 + k_2$ constraints.
6. If $e = (e_1 - e_2)$, then e is a k-bounded relation containing $\mathcal{P}(n_1, n_2, k_1, k_2)$ tuples, each of which is represented with at most $\mathcal{P}'(k_1, k_2)$ constraints. In the binary case (e_1 and e_2 binary), $\mathcal{P}(n_1, n_2, k_1, k_2) = n_1 \times (3n_2k_2)^{2k^2}$, and $\mathcal{P}'(k_1, k_2) = max(k_1, 2k^2)$. For larger arities, both the number of tuples and the number of constraints per tuple are bounded by similar polynomials.

The above follows from the definition of the algebraic operations in Section 4, from Lemma 5.4 for the case of the projection, and from Lemma 5.5 for the case of the set difference. In this last case, $(e_1 - e_2) = (e_1 \cap e_2^c)$, where e_2^c is a k-bounded relation containing, in the binary case, a maximum of $(3n_2k_2)^{2k^2}$ tuples, each having at most k^2 constraints.

It follows that the number of tuples and the number of constraints in each tuple are bounded by some polynomial function. For each algebraic sub-expression of the query Q, and each tuple t of the adequate form obtained as described above, we associate a gate $[Q, t]$ in the circuit. Other gates are also required in computing the additions of constants for the new constraints resulting from the projection operator. As shown in Lemma 5.4, there are only a polynomial amount of these. The selection also requires built-in gates to encode the constraint in the selection itself. This is easily done with a number of gates bounded by a constant in the size of the query. Essentially, no more gates are needed to encode the whole circuit. It follows that the number of gates needed to encode the whole computation is also polynomially bounded.

We now see how the wires between the gates previously presented, can be uniformly defined.

The algebraic operations have various effects on their inputs. They can modify (or rearrange) the initial tuples and/or the constraints. (i) The union operation changes only at the relation level and the initial tuples remain unchanged. (ii) Cartesian product, selection, and intersection create new tuples from old tuples, by using the initial constraints which are not changed. (iii) Set difference creates new constraints obtained from old constraints, by just changing the predicates in the constraints. (iv) Projection creates new constraints, with new parameters as shown in Lemma 5.4.

In the case of $\cup, \cap, \times$, and σ_F, it is clear that the new tuples can be computed easily in AC^0. More precisely, these operations result only in a reorganization of existing constraints inside the tuples, and of tuples inside the new relations. The wires are essentially used to copy values (with no computation). They do not have to be materialized.

Two operators, projection and set difference, deserve a more thorough examination. Indeed, they result in the definition of new constraints, with new parameters obtained from old parameters by addition, or iterated addition. It follows from Lemmas 5.4 and 5.5, that the two operations can be computed in uniform AC^0. □

In the next section, we examine consequences of the complexity upper bound on the expressive power of linear constraints.

6 Expressive Power

In this section, we study the expressive power of first order query languages for linear constraint databases. In particular we consider queries from relational database theory (parity), graph theory (graph connectivity), and geometry (region connectivity) and show that these queries are not first order expressible. The proof of these results uses the AC^0 upper bound on data complexity (Theorem 5.2) and first order reductions from boolean functions, such as PARITY which is known to be outside AC^0 [FSS84].

Let $\sigma = \{R\}$ be a signature where R is a unary relation symbol. For a database instance I of σ, the *parity* query answers "yes" if $I(R)$ is finite and has an even cardinality. The *graph connectivity* query is defined over a signature consisting of a single binary relation G. The query answers "yes" on an instance I if $I(G)$ is a connected finite graph. For the third example, we consider the *k-dimensional region connectivity* query over possibly infinite input instances, where $k \geq 1$. The query is also a boolean query and answers "yes" on an instance I if every pair of points in $I(R)$ can be linked by a continuous curve lying entirely in $I(R)$. Note that for $k = 1$, the query can be easily expressed.

The following results follow from Theorem 5.2.

Theorem 6.1 The following queries are not definable in $\{=, \leq, +\} \cup \mathbb{Q}$:

1. Parity of cardinality,
2. Graph connectivity,
3. k-dimensional region connectivity for each $k \geq 2$.

Proof: By Theorem 5.2, it is sufficient to show that these queries are not in AC^0. We first consider the parity query and describe a straightforward reduction from the boolean function PARITY. The PARITY function takes n boolean inputs and returns "true" if the number of inputs equal to 1 is even. Now let $x_1, \ldots, x_n$ be the n inputs for PARITY. We construct a database I over the signature with one unary relation symbol R as follows: $I(R) = \{i \mid x_i = 1\}$. Clearly the database is definable using only order constraints, without the addition symbol. In other words, I is in $\mathcal{K}_0$ (0-bounded). Obviously, the construction can be done in first order and $\text{PARITY}(x_1, \ldots, x_n) = 1$ iff the parity query on I answers "yes". For graph connectivity, we use the classical reduction from the parity query. Let I be an input instance of the parity query and G be a binary relation symbol. Suppose $I(R) = \{a_1, \ldots, a_n\}$, and, without loss of generality, $a_1 < a_2 < \cdots < a_n$. We define an instance J over G as follows. Let $J(G)$ be the symmetric closure of the set $\{(a_1, a_n)\} \cup \{(a_i, a_{i+2}) \mid 1 \leq i \leq n-2\}$. It is easy to verify that parity on I answers "yes" iff $J(G)$ is connected. Finally, for region connectivity in dimension $k \geq 2$, it is shown in [GS95] that it is not in AC^0, by a reduction from the boolean function MAJORITY. $\square$

References

[AHV94] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, 1994.

[Ban78] F. Bancilhon. On the completeness of query languages for relational data bases. In *Proc. 7th Symp. on Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science, pages 112–123. Springer-Verlag, 1978.

[Bea76] D. R. Bean. Recursive Euler and Hamiltonian paths. In *Proc. American Mathematical Society*, volume 55, pages 385–394, 1976.

[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.

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

[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.

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

[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. Finitely representable databases, 1995. Full version of [GS94], submitted to *JCSS*.

[Har91] D. Harel. Hamiltonian paths in infinite graphs. *Israel Journal of Mathematics*, 76:317–336, 1991.

[HH93] T. Hirst and D. Harel. Completeness results for recursive data bases. In *Proc. 12th ACM Symp. on Principles of Database Systems*, pages 244–252, 1993.

[HH94] T. Hirst and D. Harel. Recursive model theory, 1994. Draft.

[Joh90] D. Johnson. A catalog of complexity classes. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, Elsevier-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)*, April 1994. (To appear in a LNCS volume, Springer-Verlag).

[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.

[Mos57] A. Mostowski. On recursive models of formalized arithmetics. *Bulletin de l'Académie Polonaise des Sciences, III*, 5:705–710, 1957.

[Par78] J. Paredaens. On the expressive power of the relational algebra. *Information Processing Letters*, 7(2):107–111, February 1978.

[PBG94] 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.

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

[Va94] J. Väänänen. Personal communication.

[Vau60] R. L. Vaught. Sentences true in all constructive models. *Journal of Symbolic Logic*, 25(1):39–53, March 1960.