Fixed-Point Query Languages for Linear Constraint Databases

Stephan Kreutzer
LuFG Mathematische Grundlagen der Informatik, RWTH Aachen
D-52056 Aachen, Germany
kreutzer@informatik.rwth-aachen.de

ABSTRACT

We introduce a family of query languages for linear constraint databases over the reals. The languages are defined over two-sorted structures, the first sort being the real numbers and the second sort consisting of a decomposition of the input relation into regions. The languages are defined as extensions of first-order logic by transitive closure or fixed-point operators, where the fixed-point operators are defined over the set of regions only. It is shown that the query languages capture precisely the queries definable in various standard complexity classes including PTIME.

1. INTRODUCTION

Manipulating spatial data is an increasingly important part of modern database systems as spatial data plays an important role in many application areas like geographical information systems or (medical) image processing. Many of the proposed data models for spatial database systems are designed for a particular type of application. Although they allow very efficient storage and manipulation of the kind of data used in this application area, they normally fail for other types of spatial data. For example, there are very advanced geographical information systems available for two dimensional as well as for three dimensional data, but they rarely work well for both dimensions, let alone higher dimensions.

This narrowed focus is often sufficient for many applications but it has several disadvantages, one being that one needs completely different database systems for different applications.

A more general approach to spatial databases is to allow arbitrary sets as database relations, as long as they have a finite presentation in some formalism, e.g. by first-order formulae. A database framework following this approach is the framework of *constraint databases*. Constraint databases have been introduced by Kanellakis, Kuper, and Revesz [17; 16] in 1990 and since then a lot of research has been done in this area. See [19] for a very detailed study of constraint databases. In this framework the spatial relations are defined by boolean combinations of polynomial (in)equalities. Databases defined in this way are called *polynomial constraint databases* and first-order logic on these databases is usually referred to as FO+POLY. Polynomial constraint databases allow a very natural representation of spatial data. The drawback is that the evaluation of queries becomes rather expensive. Although it is known that first-order queries on polynomial constraint databases have NC data complexity, the constants involved are fairly large and the algorithms become unusable for practical applications. As semi-linear data is sufficient for most practical purposes, it is quite common to use linear approximations to the spatial data, that is to consider *linear constraint databases*, where the relations are represented by boolean combinations of linear (in)equalities. But first-order logic on these databases, usually referred to as FO+LIN, is not expressive enough as many interesting queries cannot be defined. Therefore more powerful languages are needed. There are different ways to increase the expressive power of first-order logic. In this paper we concentrate on increasing expressive power by adding recursion mechanisms. This has successfully been done for dense order constraint databases. (See [17; 12; 9]) Unfortunately linear constraint databases are not so well behaved concerning recursion. A naive definition of, e.g., least fixed-point logic leads to a non-terminating and undecidable language, as it is possible to define the natural numbers with addition and multiplication by least fixed-point logic over $(\mathbb{R}, <, +)$. (See [18] for an investigation on termination properties for query languages with recursion.) In order to guarantee termination one has to restrict the application of the fixed-point operator.

Besides termination, one encounters problems with the closure of languages over linear constraint databases - the result of a query on a linear constraint database must itself be linear again. Thus a language powerful enough to define e.g. convex hulls has to be restricted in some sense, since multiplication can easily be defined if the computation of convex hulls is possible. (See Section 4 for details.)

Attempts to define query languages with recursion mechanisms on linear constraint databases can be found in [11; 5]. Grumbach and Kuper [11] define a query language based on least fixed-point logic. Termination is guaranteed by taking the simultaneous fixed-point of two formulae, the first speaking only about the ordered set of reals whereas the second formula is allowed to use addition but is required to satisfy rather severe restrictions. It is shown that the language captures PTIME on the class of linear constraint databases but

expressing queries in this language is not very intuitive.
Geerts and Kuijpers [5] analyze properties of spatial datalog. They show that connectivity of spatial relations can be queried by a terminating spatial datalog program, but generally spatial datalog queries will not terminate on every input.

In this paper we take a different approach to define recursive query languages over linear constraint databases, by allowing fixed-point computations to range only over a finite collection of pre-defined sets. More precisely, we consider logics over two sorted structures. The first sort is the set of real numbers with order and addition defined on them. To define the second sort, we decompose the input relation into regions, which are connected subsets of the underlying space $\mathbb{R}^d$. The set of regions forms the second sort. Thus there are also two sorts of variables. The first one, called *element variables*, range over the real numbers whereas the second type of variables, called *region variables*, range over the regions. The connection between the two worlds is given by the containment predicate, stating that a point, given by a tuple of real numbers, is contained in a region. Now the languages are defined by extending first-order logic by (deterministic) transitive closure, least or partial fixed-point operators, where the operators iterate over the (finite) set of regions only. We show that many interesting queries can be formulated naturally in these languages. To estimate the exact expressive power, it is shown that exactly the boolean queries decidable in various important complexity classes can be expressed in this family of query languages.

Similar languages can be defined for polynomial constraint databases using a cell decomposition for semi-algebraic sets, but the complexity of query evaluation will be much higher.

The results presented here also contribute to a research program active for some years now, where it is investigated in how far the approach and methods of descriptive complexity theory can be applied to infinite, but finitely representable structures. Research here considers various formalisms to represent infinite structures, examples being recursive structures [14], metafinite structures and complexity theory over the reals [8; 10], and constraint databases. In this research area the addition of fixed-points to logics is still an open problem.

The paper is organized as follows. In the next section we give a precise definition of constraint databases and queries. In Section 3 we define some notions from convex geometry we need in the sequel and define a decomposition of the input relation and the underlying space. Section 4 defines the two-sorted structures and first-order logic on them. In Section 5 extensions of first-order logic by fixed-point operators are defined and some examples are given. We investigate the expressive power and complexity of these languages in the Section 6, where it is shown that the least fixed-point extension captures all boolean PTIME queries. In Section 7 logics based on transitive closure constructs are introduced together with a different decomposition of the input relation. We close with a summary and remarks on further work. Especially, we comment on extensions of the logics capable of capturing non-boolean queries.

2. PRELIMINARIES

Constraint databases. The basic idea in the definition of constraint databases is to allow infinite relations which have a finite presentation by a quantifier-free formula. Let $\mathfrak{A}$ be a τ-structure, called the *context structure*, and $\varphi(x_1, \dots, x_n)$ be a quantifier-free formula of vocabulary τ. We say that a n-ary relation $R \subseteq A^n$ is *represented by* $\varphi(x_1, \dots, x_n)$ *over* $\mathfrak{A}$ iff R equals $\{\bar{a} \in A^n : \mathfrak{A} \models \varphi[\bar{a}]\}$. Let $\sigma := \{R_1, \dots, R_k\}$ be a relational signature. A σ-constraint database over the context structure $\mathfrak{A}$ is a σ-expansion $\mathfrak{B} := (\mathfrak{A}, R_1, \dots, R_k)$ of $\mathfrak{A}$ where all R_i are finitely represented by formulae φ_{R_i} over $\mathfrak{A}$. The set $\Phi := \{\varphi_{R_1}, \dots, \varphi_{R_k}\}$ is called a finite representation of $\mathfrak{B}$.

By definition, constraint databases are expansions of a context structure by finitely representable database relations. Note that the same relation can be represented in different ways, e.g. $\varphi_1 := 0 < x \wedge x < 10$ and $\varphi_2 := (0 < x \wedge x < 6) \vee (6 < x \wedge x < 10) \vee x = 6$ are different formulae defining the same relation. Two representations Φ and Φ' are $\mathfrak{A}$-*equivalent*, iff they represent the same database over $\mathfrak{A}$.

To measure the complexity of algorithms taking constraint databases as inputs we have to define the size of a constraint database. Unlike finite databases, the size of constraint databases cannot be given in terms of the number of elements stored in them but has to be based on a representation of the database. Note that equivalent representations of a database need not be of the same size. Thus the size of a constraint database cannot be defined independent of a particular representation. In the following, whenever we speak of a constraint database $\mathfrak{B}$, we have a particular representation Φ of $\mathfrak{B}$ in mind. The size $|\mathfrak{B}|$ of $\mathfrak{B}$ is then defined as the sum of the length of the formulae in Φ. This corresponds to the standard encoding of constraint databases by the formulae of their representation.

In this paper we consider linear constraint databases, that is databases defined by boolean combinations of linear (in)equalities. In the literature one can find two models of computation for such databases. The first model allows real coefficients in the formulae. The computational devices used can store any real number in a single storage cell and have built in functions like addition and multiplication. The second model allows only rational coefficients in the formulae. Since one can multiply the atoms in the formulae with the least common denominator, this is equivalent to allowing integer coefficients only. The integers are stored bitwise on the Turing tape. We take the second approach and consider the context structure $\mathfrak{A} := (\mathbb{R}, <, +)$. The formulae are allowed to use elements from $\mathbb{Z}$ as parameters. As usual we require the formulae representing the input relations to be in disjunctive normal form and consider databases with one single spatial relation only. This restriction is not crucial but helps to simplify the presentation. For notational reasons we disallow the use of negation, but allow $\leq, >$, and $\geq$ instead. Clearly, $\neq$ can be defined using $<$.

Constraint queries. Fix a context structure $\mathfrak{A}$. A constraint query is a mapping Q from constraint databases over $\mathfrak{A}$ to finitely representable relations over $\mathfrak{A}$. Note that queries are abstract, i.e. they depend only on the database not on their representation. That is, any algorithm that computes Q, taking a representation Φ of a database $\mathfrak{B}$ as input and producing a representation of $Q(\mathfrak{B})$ as output, has to compute on two $\mathfrak{A}$-equivalent representations Φ and Φ' output formulae that are not necessarily the same, but represent the same relation on $\mathfrak{A}$.

In the sequel we are particularly interested in queries defined by formulae of a given logic $\mathcal{L}$. Let $\varphi \in \mathcal{L}$ be a formula with k free variables. Then φ defines the query Q_φ mapping a constraint database $\mathfrak{B}$ over $\mathfrak{A}$ to the set

$$\varphi^{\mathfrak{B}} := \{(a_1, \ldots, a_k) \ : \ \mathfrak{B} \models \varphi[\bar{a}]\}.$$

In order for Q_φ to be well defined, this set must be representable by a quantifier-free formula. If φ is first-order, this means that $\mathfrak{A}$ admits quantifier elimination. For more powerful logics than first-order logic, the additional operators must be eliminated as well. A logic $\mathcal{L}$ is *closed* for a class $\mathcal{C}$ of constraint databases over $\mathfrak{A}$, if for every $\varphi \in \mathcal{L}$ and every $\mathfrak{B} \in \mathcal{C}$ the set $\varphi^{\mathfrak{B}}$ can be defined by a quantifier-free first-order formula over $\mathfrak{A}$.

Typical questions that arise when dealing with constraint query languages are the complexity of query evaluation for a certain constraint query language and the definability of a query in a given language. For a fixed query formula $\varphi \in \mathcal{L}$, the *data complexity* of the query Q_φ is defined as the amount of resources (e.g. time, space, or number of processors) needed to evaluate the function that takes a representation Φ of a database $\mathfrak{B}$ to a representation of the answer relation $Q_\varphi(\mathfrak{B})$.

3. ARRANGEMENTS

In this section we explain some notions from convex geometry that will be needed later. The presentation mostly follows [4]. See also [6; 23] for details.

Let $P \subseteq \mathbb{R}^d$ be a set of points in $\mathbb{R}^d$. The *affine support* or *affine hull* of P is defined as the smallest affine subspace of $\mathbb{R}^d$ containing P. The *convex hull of* P is defined as

$$conv(P) := \{x \ : \exists p_1, \ldots, p_n \in P, n \in \mathbb{N}, \exists a_1, \ldots a_n \in \mathbb{R}^{\geq 0}, \Sigma a_i p_i = x, \ \Sigma a_i = 1\}.$$

We define the *open convex hull of* P as the interior of $conv(P)$ with respect to its affine support. It can be defined as

$$openconv(P) := \{x \ : \exists p_1, \ldots, p_n \in P, n \in \mathbb{N}, \exists a_1, \ldots a_n \in \mathbb{R}, a_i > 0, \ \Sigma a_i p_i = x, \ \Sigma a_i = 1\}.$$

Whenever we speak about the interior of a set or a set being open, we always mean with respect to its affine support.

Usually, a polyhedron in $\mathbb{R}^d$ is defined as the intersection of finitely many *closed* halfspaces in $\mathbb{R}^d$. For our purposes it is more convenient to allow the intersection with *open* halfspaces as well. Thus we define a *polyhedron in* $\mathbb{R}^d$ as the intersection of finitely many open or closed halfspaces. It is *bounded*, if it is entirely contained in some d-dimensional hypercube of edge length l for some $l \in \mathbb{R}^{\geq 0}$. A bounded polyhedron is called a *polytope*.

Recall that a linear constraint relation S is defined by a formula $\varphi_S := \bigvee_i \bigwedge_j \varphi_{ij}$, where each φ_{ij} is a linear inequality, defining a halfspace, or a linear equation, defining a hyperplane.

Thus, each conjunct in $\varphi_i := \bigwedge \varphi_{ij}$ defines a polyhedron and the relation S consists of a union of polyhedra. Let $\mathfrak{G}(S) := \{g_1, \ldots, g_n\}$ be the set of (in)equalities occurring in φ_S. Consider the set of hyperplanes

$$\mathfrak{H}(S) := \{h \ : \ \text{there is an equation } g \in \mathfrak{G}(S) \text{ and } h = g, \text{ or } g \text{ is an inequality and } h \text{ is obtained by replacing in } g \text{ inequality by equality } \}.$$

See Figure 1 and 2 for a spatial relation and the corresponding set of hyperplanes.

For each $h := \Sigma a_i x_i = b \in \mathfrak{H}(S)$ we define the set of points being *above, on*, or *below* h, where a point $p := (p_1, \ldots, p_d)$ is above h if $\Sigma a_i p_i > b$, on h if $\Sigma a_i p_i = b$, and below h if $\Sigma a_i p_i < b$. We denote by h^- the set of points below h and by h^+ the set of points above h. Clearly, any two points on the same side of all hyperplanes in $\mathfrak{H}(S)$ are either both contained in or both not contained in S. To see this, recall that the hyperplanes in $\mathfrak{H}(S)$ arise from the atoms in φ_S. Points on the same side of a hyperplane h cannot be separated by the corresponding atom. Clearly, if the points cannot be separated by the atoms in φ_S, then the boolean combination of the atoms cannot separate them either.

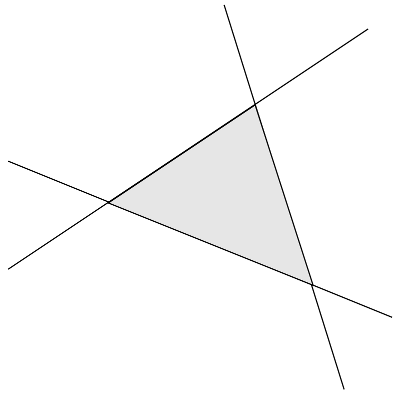

Figure 1: Example of a spatial relation S.

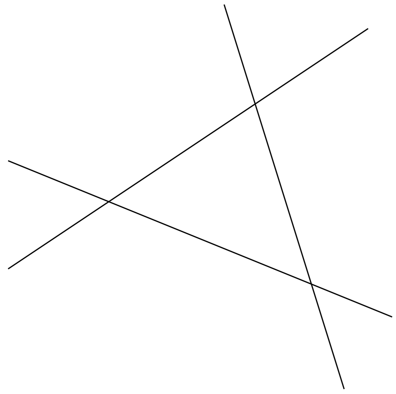

Figure 2: Set of hyperplanes induced by S.

More precisely, let p be a point in $\mathbb{R}^d$. The *position* $v_i(p)$ with respect to $h_i \in \mathfrak{H}(S)$ is defined as

$$v_i(p) := \begin{cases} +1 & \text{if } p \in h_i^+ \\ 0 & \text{if } p \in h_i \\ -1 & \text{if } p \in h_i^-. \end{cases}$$

The position of a point with respect to $\mathfrak{H}(S)$ is determined by the vector $(v_1(p), \ldots, v_n(p))$, called the *position vector of* p, and any two points sharing the same position vector are inseparable by φ_S.

We call a set of all points sharing the same position vector a *face* and the dissection of $\mathbb{R}^d$ induced by the set $\mathfrak{H}(S)$ of hyperplanes an *arrangement* $\mathcal{A}(S)$. This dissection of $\mathbb{R}^d$ into faces is a partition of $\mathbb{R}^d$ with the property that every face is either contained in or disjoint to S. See Figure 3 for the decomposition of the database shown above.

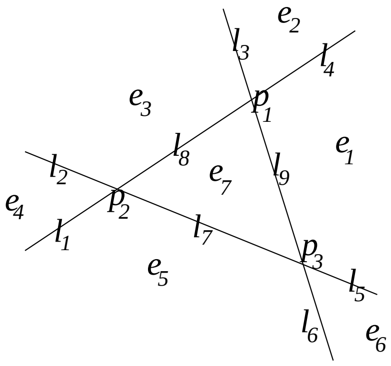

Figure 3: Arrangement $\mathcal{A}(S)$.

The *dimension of a face* in the decomposition is defined as the dimension of its affine support. Thus we have seven 2-dimensional faces e_1 to e_7, nine 1-dimensional faces l_1 to l_9, and three 0-dimensional faces p_1 to p_3. As usual, we call 0-dimensional faces *vertices*.
We say, that two faces f and g are *incident*, if one is of dimension one less than the other and it is contained in the boundary of the other region.
Typically, arrangements are stored in a data structure like the *incidence graph*. The incidence graph contains a vertex for each face in the arrangement as well as two additional vertices, one representing a virtual (-1)-dimensional face, denoted by $\varnothing$, which is incident to every 0-dimensional face, and one vertex representing a $(d+1)$-dimensional face, written as $\mathcal{A}(S)$, where every d-dimensional face is incident to. We call the last two vertices *improper* and the other *proper vertices*. Each proper vertex v stores the position vector of the points contained in the corresponding face f as well as two lists of directed edges, one containing edges pointing at the vertices whose faces are incident to f and one containing edges pointing at the vertices to whose faces f is incident. Figure 4 shows the incidence graph for the part of the arrangement of Figure 3 containing the faces around p_2.

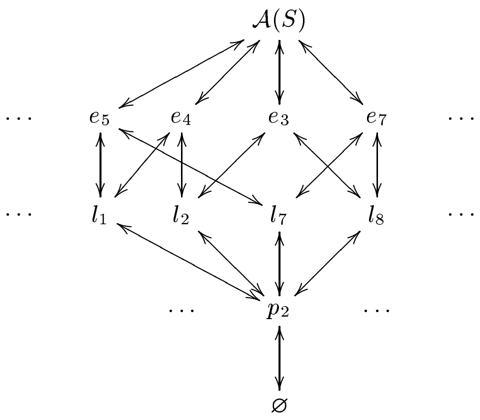

Figure 4: Part of the incidence graph for $\mathcal{A}(S)$.

As every vertex stores the position vector of the corresponding face, a conjunction of atoms defining the face can easily be obtained from $\mathfrak{H}(S)$ and the incidence graph for $\mathcal{A}(S)$. Also the incidence relation can be efficiently decided.
It is known, that an arrangement for a set of n hyperplanes in $\mathbb{R}^d$ can be computed in time $\mathcal{O}(n^d)$. See Theorem 7.6 in [4]. Besides the known sequential algorithms, there have some attempts been made to define parallel algorithms for computing arrangements [1; 7], leading to algorithms running in parallel time $\mathcal{O}(\log n)$ using a polynomial number of processors on a CREW PRAM (See [7]).
Since the number of hyperplanes in $\mathfrak{H}(S)$ is always less or equal the number of atoms in the representation of the database, the following theorem is immediate.

Theorem 3.1. *Let $\mathfrak{B} := ((\mathbb{R}, <, +), S)$ be a linear constraint database. The arrangement $\mathcal{A}(S)$ can be computed in polynomial time with respect to the size of the representation of $\mathfrak{B}$.*

4. FIRST-ORDER LOGIC WITH REGION VARIABLES

In this section we use the arrangement defined above to introduce *first-order logic with region variables*, which is first-order logic on certain two-sorted structures. We first introduce the structures the logics operate on.
Let $\mathfrak{B}$ be a database with a d-ary spatial relation S. The definition of the structures is based on a dissection of $\mathbb{R}^d$ into regions, where a region is a connected subset of $\mathbb{R}^d$. We denote the set of regions by *region*(S). To define first-order logic with region variables and its extension by fixed-point operators, we use the arrangement $\mathcal{A}(S)$ as dissection. Thus, here and in the next section *regions*(S) is defined as the set of faces in $\mathcal{A}(S)$. In Section 7 we will use a different decomposition of $\mathbb{R}^d$ into regions to define a logic based on a transitive closure operator.

Definition 4.1. *Let $\mathfrak{B} := ((\mathbb{R}, <, +), S)$ be a linear constraint database, where S is a d-ary relation symbol.
The structure $\mathfrak{B}$ gives rise to a two-sorted structure $\mathfrak{B}^{Reg} := (\mathbb{R}, Reg; \leq, +, S, adj, \in)$, called the* region extension *of $\mathfrak{B}$, with sorts $\mathbb{R}$ and $Reg := regions(S)$, the adjacency relation $adj \subseteq Reg \times Reg$, where two regions are adjacent if there is a point p in one of them such that every ε-neighbourhood of p has a non-empty intersection with the other region, and the element containment relation $\in$ between elements from $\mathbb{R}^d$ and Reg, where $\bar{x} \in R$ is true iff the point $\bar{x}$ is contained in the region R.*

The ε-neighbourhood of a point $p = (p_1, \dots, p_d) \in \mathbb{R}^d$ can be defined as the set $\{p' := (p'_1, \dots, p'_d) \; : \; |p_i - p'_i| < \varepsilon \text{ for all } 1 \leq i \leq d\}$. It follows immediately that the adjacency relation is first-order definable. Thus adding it to the signature of $\mathfrak{B}^{Reg}$ is a mere convenience.
The above definition of adjacency is equivalent to an alternative definition, where two regions are adjacent iff one region is contained in the closure of the other. Thus if two regions are adjacent, then one is of strictly lower dimension than the other. Further, any two regions being incident are adjacent too.

Definition 4.2. *Let $\mathcal{V}$ be a set of* element variables *and $\mathcal{R}$ be a set of* region variables. *We define RegFO as first-order logic over structures $\mathfrak{B}^{Reg}$. Thus we have quantifiers $\forall x, \exists x$ for element variables ranging over the reals and $\forall R, \exists R$ for region variables ranging over the set Reg of regions. A RegFO query is a query defined by a RegFO formula without free region variables.*

Usually we use small letters $x, y, z, \ldots$ for element variables and capital letters $R, X, Y, \ldots$ to denote region variables. Clearly, every linear constraint database has an unique region extension. Therefore we freely speak about a database $\mathfrak{B}$ being a model of a *RegFO*-sentence instead of explicitly mentioning its region extension $\mathfrak{B}^{Reg}$. We also use the database as input for algorithms and Turing machines.

THEOREM 4.3. *Every RegFO query on linear constraint databases has polynomial time data complexity.*

PROOF. To prove the theorem we have to show that for each *RegFO*-query φ there is a PTIME algorithm which, given (a representation of) a database $\mathfrak{B}$, computes a quantifier-free first-order formula defining $\varphi^{\mathfrak{B}}$.
By Theorem 3.1 we know that the arrangement of the input database can be computed in time polynomial in the size of the database. The proof of the theorem now follows by induction on the structure of the query. The first-order cases follow immediately from the PTIME complexity of first-order queries on linear constraint databases as proved in [17] and [13]. The case of atoms built up by the adjacency relation is trivial as the relation is first-order definable.
The remaining cases are where region variables occur. Computing a formula defining a region can be done in polynomial time, since the incidence graph stores the position vector of the contained points for every region. Once the position vector is given the construction of the formula is trivial.
The case of a formula of the type $\exists P \varphi(P)$, where P is a region variable, can be handled as follows. For each region R we construct the formula defining R, replace the occurrences of P in φ by this formula and evaluate the resulting formula. By induction, this can be done in polynomial time. The output now consists of the disjunction of the formulae for each region.
The universal quantification can be dealt with analogously, taking the conjunction instead of disjunction. □

A natural question arising is whether it is necessary to restrict the region variables to regions of the input relation instead of introducing quantifiers of the kind $\exists R \in region(\psi)$ meaning that R is a region variable ranging over the set of regions of the relation defined by ψ. Unfortunately this makes the logic too expressive, as convex closure and thus multiplication can easily be defined. The definition of multiplication by convex closure is demonstrated in Figure 5.

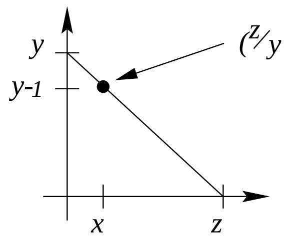

Figure 5: Defining multiplication by convex closure.

The relation $mult(x, y, z)$, true for x, y, and z if $x \cdot y = z$, can be defined as follows. We assume w.l.o.g. that x, y, and z are positive. Consider the points $(0, y)$ and $(z, 0)$ in the plane. The convex closure of these points is the line segment as shown in Figure 5. The point on the line segment having $(y-1)$ as second coordinate has the first coordinate $\frac{z}{y}$. Thus, if $(x, y-1) \in conv(\{(0, y), (z, 0)\})$, then $x = \frac{z}{y}$ and $x \cdot y = z$. Having defined first-order logic over region extensions we now turn to more expressive languages.

5. LOGICS BASED ON FIXED-POINT OPERATORS

We now consider extensions of *RegFO* by fixed-point constructs. First, we give a definition of a language based on least fixed-point induction. Logics based on transitive closure operators will be defined later. In all of these logics, the fixed-point induction is allowed only on the region domain, guaranteeing closure of the queries and efficiency of their evaluation.

DEFINITION 5.1. *Let $\mathcal{M} := \bigcup_{i \in \mathbb{N}} \mathcal{M}_i$ be a set of variables, called* set variables, *where each $\mathcal{M}_i$ is a countable infinite set of variables of arity i. These variables will be interpreted as sets of regions.*
Define RegLFP as RegFO extended by the following rules.

- *If $M \in \mathcal{M}$ is a set variable of arity k and $R_1, \ldots, R_k$ are region variables, then*
$$\psi := MR_1 \ldots R_k \in RegLFP$$
and $\mathrm{free}(\psi) := \{M, R_1, \ldots, R_k\}$.
- *Let $M \in \mathcal{M}$ be a set variable of arity k and $R_1, \ldots, R_k$, $X_1, \ldots, X_k$ be region variables. Further, let φ be a RegLFP-formula with* $\mathrm{free}(\varphi) = \{M, X_1, \ldots, X_k\}$ *such that φ is positive in M. Then*
$$[LFP_{M, X_1, \ldots, X_k} \varphi](R_1, \ldots, R_k) \in RegLFP$$
with free variables $\{R_1, \ldots, R_k\}$.
- *If φ is a formula with exactly one free element variable but arbitrarily many region variables $\bar{P}$, then*
$$\psi := [rBIT\ \varphi](R_n, R_d) \in RegLFP,$$
where R_d, R_n are region variables. The free variables of ψ are R_d, R_n and $\bar{P}$.

A RegLFP query is a query defined by a RegLFP formula without free region or set variables.
The logics RegIFP and RegPFP are defined analogously by using inflationary or partial fixed-point operators instead.

Let $\mathfrak{B} := ((\mathbb{R}, <, +), S)$ be a database. The semantics of the LFP operator is the standard least fixed-point semantics on the region domain.
Let $\bar{X} := X_1, \ldots, X_n$ be a sequence of region variables, M be a set variable of arity n, and $\varphi(M, \bar{X})$ a formula with free variables M and $\bar{X}$. φ gives rise to function f_φ given by

$$f_\varphi : \begin{array}{ccc} \mathcal{P}(Reg^n) & \to & \mathcal{P}(Reg^n) \\ M & \mapsto & \{\bar{P} \in Reg^n \ : \ \mathfrak{B} \models \varphi[M, \bar{P}]\}. \end{array}$$

If φ is positive in M, then f_φ is monoton and the least fixed-point of f_φ is guaranteed to exist.
Now consider $\psi := [LFP_{M,\bar{X}} \varphi(M, \bar{X})](\bar{X})$. $\mathfrak{B} \models \psi[\bar{P}]$ for a tuple of regions $\bar{P}$ iff $\bar{P}$ is contained in the least fixed-point of f_φ. See [3] for a detailed introduction to least fixed-point logics.

The rBIT operator requires some explanation. Consider the set of regions with dimension 0 and let n be the number of regions in it. This set can be ordered by the ordering induced by the lexicographical order on the points they contain. Clearly, this is a total order. Now let $\varphi(x, \bar{P})$ be a formula with one free element variable x and arbitrary many free region variables $\bar{P}$. If, for a given interpretation of the free region variables $\bar{P}$, the formula φ is satisfied by exactly one rational number a, then $[\mathrm{rBIT}\ \varphi](R_n, R_d)$ is true for a pair of regions R_i, R_j, if

1. they are 0-dimensional and the i-th bit of the bit representation of a's numerator is 1 and the j-th bit of the bit representation of a's denominator is 1, or
2. $a = 0$, $R_i = R_j$ and both are higher dimensional regions.

Otherwise $[\mathrm{rBIT}\ \varphi]$ defines the empty set.
The main ingredient of the language is the least fixed-point operator. The rBIT operator should be considered a technical neccessity which is needed to prove that *RegLFP* captures PTIME on linear constraint databases.
Before proving complexity bounds for *RegLFP*-queries, we give some examples of queries expressible in *RegLFP*.
The first example is connectivity, which plays an important role in spatial databases. Let S be d-ary. Connectivity can be defined by the query

$$\begin{aligned} Conn := \forall\bar{x}\forall\bar{y}(S\bar{x} \wedge S\bar{y} \rightarrow (\exists R_x \exists R_y\ \bar{x} \in R_x \wedge \bar{y} \in R_y \wedge \\ [LFP_{M,R,R'}((R = R' \wedge R \subseteq S) \vee (\exists Z\ M(R, Z) \\ \wedge\ adj(Z, R') \wedge R' \subseteq S))](R_x, R_y))) \end{aligned}$$

stating that for each pair of points $\bar{x}, \bar{y} \in S$, the regions R_x and R_y they are contained in, can be connected by a sequence of adjacent regions contained in S.
We now turn to a less technical example as it might occur in applications of GIS systems. Figure 6 gives an (abstract) map of a country's border, a river, and some cities on its bank. Since we don't consider "mixed-part" databases, databases with a standard relational database part in addition to the spatial information, we assume that the information whether a point belongs to the border, a city, or the river is stored in the third dimension. Thus we take formulae *river*(R) and *city*(R) for granted, stating that a region R belongs to a river or city respectively.
Now imagine that the river is polluted by certain chemicals at some cities on the river bank and we ask whether there is a part of the river, where a particular combination of chemicals can be found. In *RegLFP* it is now an easy task to start at the spring and follow the river collecting the chemicals the river is polluted with. Once we have seen the desired combination of chemicals, we mark the city. This can be expressed in *RegLFP* as follows, using formulae *spring*, $chem_1$, $chem_2$ with the obvious semantics.

$$\begin{aligned} \psi := &\exists R_1 \exists R_2\ R_1 \neq R_2 \wedge \\ &[LFP_{M,R,R'}((spring(R) \wedge R = R') \\ &\vee (\exists Z \exists Z'\ M(Z, Z') \wedge river(R) \wedge adj(Z, R) \wedge R = R') \\ &\vee (\exists Z \exists Z'\ M(Z, Z') \wedge chem_1(Z) \wedge chem_2(R) \wedge R' = Z) \\ &](R, R') \end{aligned}$$

The first two disjuncts in the formula inside the LFP operator make sure that, beginning at the spring and following the course of the river, at each induction step a new pair (R, R), for a region R of the river, is added to M. The

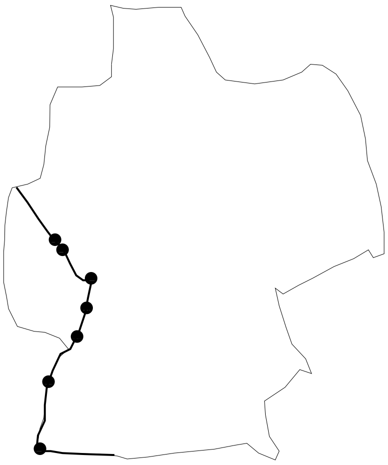

Figure 6: Map showing a river and some cities on its bank.

last disjunct checks that whenever there is a part of the river with the second chemical, whether there was a part with the first chemical before. Thus, the fixed-point contains only pairs (R, R) of equal regions, except for when the desired combination of chemicals is found.

6. COMPLEXITY

In this section we investigate the expressive power and complexity of *RegLFP*-queries. The main theorem states that *RegLFP* captures PTIME in the usual sense of finite model theory, meaning that the class of properties of linear constraint databases definable in *RegLFP* is exactly the class of properties decidable in PTIME. In other words, *RegLFP* can express all PTIME computable *boolean* queries, but it falls short of being able to express all PTIME queries of higher arity. See Section 8 for comments on non-boolean queries.
First we show the PTIME complexity of *RegLFP*-queries.

THEOREM 6.1. *The data-complexity of RegLFP-queries is* PTIME.

PROOF. Again the proof of the theorem is by induction on the structure of the *RegLFP*-query. The *RegFO* cases are already handled by Theorem 4.3. The fixed-point cases are trivial since the fixed-point induction is only defined for the finite set of regions.
The only case requiring some care is the rBIT operator. Let $\varphi(x, \bar{P})$ be a *RegLFP*-formula. Clearly, for any given interpretation of the free region variables $\bar{P}$, the evaluation of $[\mathrm{rBIT}\ \varphi]$ can be done in polynomial time. Since there are only polynomially many different interpretations of the free region variables the evaluation of the operator can be done in PTIME as well. □

We now state the main theorem of this section. Since we require some technical restrictions on the databases, we first state precisely what we understand by a logic capturing a complexity class.

DEFINITION 6.2. *A linear constraint database $\mathfrak{B}$ has the* small coordinate property *if the absolute values of the coordinates of all points contained in a 0-dimensional region are bounded by $2^{O(n)}$, where n is the number of regions in the region extension of $\mathfrak{B}$.*

We now precisely define the notion of capturing we use here.

DEFINITION 6.3. *We say that a logic $\mathcal{L}$ captures a complexity class $\mathcal{C}$ on a class $\mathcal{K}$ of linear constraint databases, if for every boolean query Q on $\mathcal{K}$ which can be decided in $\mathcal{C}$, there is a sentence $\varphi \in \mathcal{L}$ such that for all linear constraint databases $\mathfrak{B} \in \mathcal{K}$*

$$Q(\mathfrak{B}) \textit{ is true iff } \mathfrak{B} \models \varphi.$$

With this notion of capturing we can state the main theorem of this section.

THEOREM 6.4. *RegLFP captures* PTIME *on the class of linear constraint databases having the small coordinate property. Analogously, RegIFP and RegPFP capture* PTIME *and* PSPACE *respectively.*

PROOF. We first prove the theorem for *RegLFP*. By Theorem 6.1 we know that *RegLFP* $\subseteq$ PTIME.
To prove the other direction we show that every class $\mathcal{C}$ of databases decidable in PTIME is definable by a *RegLFP* sentence φ. Since $\mathcal{C} \in$ PTIME, there is a polynomial time Turing machine M deciding $\mathcal{C}$. Thus it suffices to show that there is a sentence $\varphi_M \in$ *RegLFP* such that a database $\mathfrak{B}$ is accepted by M iff $\mathfrak{B} \models \varphi_M$.
Letting logical formulae encode runs of Turing machines is a standard procedure in descriptive complexity theory. See [3; 15] for more details. One can define the sentence φ_M as a conjunction $\varphi_M :=$ START $\wedge$ COMPUTE $\wedge$ END, where START is a formula coding the start configuration, COMPUTE codes the transition function, and END states that M never reaches a rejecting state. The formula START depends on a formula β defining the input tape of M, i.e. the representation of $\mathfrak{B}$. The definition of β is the only essential difference in the case of linear constraint databases. The other formulae can be defined as in the finite case, using region variables ranging over the set of 0-dimensional regions instead of first-order variables.
We have to show that there is a formula β defining the representation of $\mathfrak{B}$. The database is represented in the following way. First we partition the set of regions into *bounded* and *unbounded* regions, where a region is bounded if it is contained in a d-dimensional hypercube of edge length l for some $l \in \mathbb{R}$. Otherwise the region is unbounded. Note that by the following rules we can define an order relation on the bounded regions. If R, R' are bounded regions and R' is of higher dimension than R, then $R < R'$. The 0-dimensional regions can be ordered by the lexicographical order on the coordinates of the points they contain. Without giving details we state that an ordering of the i-dimensional regions can be defined from the lexicographical ordering of all $(i+1)$-tuples of 0-dimensional regions.
By results of [21; 22; 2] the dimension of a region is first-order definable. Consequently the order relation is *RegFO*-definable.
We now define the representation of the bounded regions. The representation is illustrated in the following figure.

$$\underbrace{\overbrace{|p_1^1|p_2^1|\dots|p_d^1|c_1|}^{R_1}\#|\dots|\#\overbrace{|p_1^n|p_2^n|\dots|p_d^n|c_n|}^{R_n}\#}_{\text{Dimension 0}}$$
$$\underbrace{|d_1^1|d_2^1|\dots|d_{n^2}^1|}_{\text{Dimension 1}}\dots\underbrace{|d_1^d|d_2^d|\dots|d_{n^{d+1}}^d|}_{\text{Dimension } d}$$

Here R_i is the ith smallest 0-dimensional region, p_j^i is the jth coordinate of the point $p_i \in R_i$ and $c_i = 1$ if and only if $p_i \in S$. Further, $d_j^i = 1$ iff the j-th i-dimensional region is contained in S. The coordinates p_j^i of the 0-dimensional regions are represented in binary (not shown in the figure.) The binary representation can be defined using the rBIT operator. The operator transforms the coordinates of the point into sequences of 0-dimensional regions representing the coordinates. As there are only n 0-dimensional regions, only coordinates up to $2^{\mathcal{O}(n)}$ can be represented in this way. Thus the databases which can be represented must satisfy the small coordinate property.
For unbounded regions we first define an ordering on the set of all unbounded regions. Let R and R' be unbounded. If R is of lower dimension than R' then $R < R'$. Otherwise let i be the dimension of R and R'. If $i = 1$ we can order the regions as follows. Note that every unbounded 1-dimensional region R has exactly one 0-dimensional region $R_0 = \{p\}$ adjacent to but not contained in it. Starting from p we can compute some arbitrary other point $q \in R$. Now the lexicographical order on (p, q) induces an order relation on the set of 1-dimensional unbounded regions. The higher dimensional unbounded regions can now be ordered analogously to the bounded case by enumerating tuples of 1-dimensional regions. Using this order relation we can represent the unbounded regions as follows.

$$\underbrace{\overbrace{|p_1|q_1|c_1|}^{R_1}\#\dots\#\overbrace{|p_m|q_m|c_m|}^{R_m}}_{\text{Dimension 1}}$$
$$\#\underbrace{|d_1^2|d_2^2|\dots|d_{m^2}^2|}_{\text{Dimension 2}}\dots\underbrace{|d_1^d|d_2^d|\dots|d_{m^d}^d|}_{\text{Dimension } d}$$

All R_i and p_i, q_i, c_i, and d_j^i are defined similar to the bounded case. By a rather technical but not very complex argument it can be shown that this representation of the database can be defined in *RegFO*.
Clearly, if there is a polynomial-time Turing machine M defining $\mathcal{C}$ using the standard encoding of constraint databases, then there also is a polynomial-time machine M' deciding $\mathcal{C}$ which uses the above representation. Thus for all databases $\mathfrak{B}$ we have

$$\mathfrak{B} \in \mathcal{C} \quad \text{iff} \quad \mathfrak{B} \text{ is accepted by } M' \quad \text{iff} \quad \mathfrak{B} \models \varphi_{M'}.$$

This concludes the proof for *RegLFP*. The proofs for *RegIFP* and *RegPFP* are analogous. $\square$

7. LOGICS BASED ON TRANSITIVE CLOSURE OPERATORS

In this section we consider extensions of *RegFO* by induction mechanisms based on transitive closure computation. In particular, we define *RegTC* and *RegDTC*, the extensions of *RegFO* by transitive and deterministic transitive closure operators. Here we encounter the problem, that Theorem 3.1 only yields a PTIME upper complexity bound for the computation of arrangements. As explained in Section 3, the arrangements can be computed in logarithmic parallel time on a CREW PRAM, and thus in AC^1, but it is not known, whether this can be reduced to NC^1 or LOGSPACE. Thus, showing that, e.g., *RegDTC*-queries have LOGSPACE data-complexity already fails at the complexity of the arrangement computation.

To deal with this problem, we define a different decomposition as in Section 3, which can be computed in NC^1.
Let $\mathfrak{B} := ((\mathbb{R}, <, +), S)$ be a linear constraint database, where S is a d-ary relation represented by $\varphi_S := \bigvee_i \psi_i$, $\psi_i := \bigwedge_j \varphi_{ij}$ and each φ_{ij} is a linear (in)equality. First, we define sets *regions*(ψ_i) separately for each disjunct ψ_i. *regions*(S) then consists of the union of *regions*(ψ_i) for all i. We roughly sketch the definition of the regions by giving an example of the computation for a polytope in the plane. See Figure 7. The precise definition covering also the cases of unbounded polyhedra and polyhedra in higher dimension can be found in Appendix A.
Consider the polytope P shown in Figure 7.

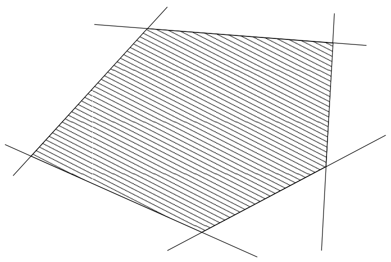

Figure 7: Polytope P defined by a disjunct in φ_S.

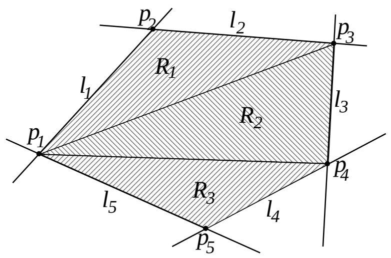

Figure 8: Decomposition of the polyhedron in Figure 7.

The regions for P are defined as follows. First, the vertices $p_1, \ldots, p_5$ of P are computed and every vertex is a region in *regions*(P). These are the 0-dimensional regions. Further, for any two vertices p, p' let $C_{p,p'}$ be the open convex closure of p and p'. If the intersection of $C_{p,p'}$ and the interior of P is empty, then $C_{p,p}$ is added as a 1-dimensional region to *regions*(P). In the example, the 1-dimensional regions are exactly the line segments $l_1, \ldots, l_5$ on the boundary of P.
Finally, we add regions in the interior of P. First, we choose the point p_1 as basis for the regions. The regions then are defined as indicated in Figure 8. The regions $R_1, \ldots, R_3$ are defined as the open convex hulls of the three bounding vertices. Further, we the line segments between p_1 and p_3 and p_1 and p_4 resp. are added to *regions*(P).
In the case of polytopes in higher dimension d, the regions in the interior would be defined by the open convex hull of $d + 1$ vertices and the regions in the boundary by the open convex hull of up to d vertices.
The regions thus defined can be computed efficiently, in NC^1. See Lemma A.1 in the appendix. We are now ready to define extensions of *RegFO* by (determinstic) transitive closure operators.

NOTE 7.1. *We now base the definition of the region extension $\mathfrak{B}^{Reg}$ of a linear constraint database $\mathfrak{B}$ on the set of regions defined above. Thus, if $\mathfrak{B} := ((\mathbb{R}, <, +), S)$ is a linear constraint database, then its region extension $\mathfrak{B}^{Reg} := (\mathbb{R}, Reg; \leq, +, S; adj, \in)$ is a two-sorted structure, where the second sort ist defined as $Reg :=$ regions(S). The definition of the relations* adj *and $\in$ is analogous to Definition 4.1.*

Recall that *regions*(S) consist of the union of regions separately defined for each disjunct ψ_i in φ_S. Regions for different polyhedra may overlap and thus the regions in *regions*(S) are not guaranteed to be either contained in or disjoint from S. Also the regions do not cover all of the underlying space $\mathbb{R}^d$. Both makes the logics less intuitive to use. Thus it would be interesting to have a decomposition with the nice properties of arrangements but computable in NC^1.

DEFINITION 7.2. *RegTC is the extension of RegFO by a transitive closure operator on the region domain. Precisely, RegTC extends RegFO by the following rule. If $\varphi \in$ RegTC and $\bar{R} := R_1, \ldots, R_m, \bar{R}' := R'_1, \ldots, R'_m$ are sequences of region variables such that* $\mathrm{free}(\varphi) = \{\bar{R}, \bar{R}'\}$ *then*

$$\psi := [TC_{\bar{R},\bar{R}'}\varphi](\bar{X}, \bar{Y}) \in openRegTC$$

and $\mathrm{free}(\psi) := \{\bar{X}, \bar{Y}\}$*, where $\bar{X}, \bar{Y}$ are sequences of m region variables each. As before, we define RegTC queries as the set of queries defined by RegTC formulae without free region variables.*
RegDTC is defined analogously using a deterministic transitive closure operator instead.

The TC operator has the standard semantics, i.e. given a formula $\psi := [TC_{\bar{R},\bar{R}'}\varphi](\bar{X}, \bar{Y})$, a database $\mathfrak{B}$, and tuples of regions $\bar{P}, \bar{P}'$, $\mathfrak{B} \models \psi[\bar{P}, \bar{P}']$ iff there is a sequence $\bar{Z}_1, \ldots, \bar{Z}_n$ of tuples of regions, $\bar{Z}_1 = \bar{X}, \bar{Z}_n = \bar{Y}$ and for all $1 \leq i \leq n-1$ $\varphi(\bar{Z}_i, \bar{Z}_{i+1})$. See [3] for a detailed introduction to transitive closure logics.
By induction on the structure of the queries one can easily show the following theorem.

THEOREM 7.3. *The data-complexity of RegTC-queries is* NLOGSPACE*. Analogously, the data-complexity of RegDTC-queries is* LOGSPACE.

In analogy to the proof of Theorem 6.1 the following theorem can be proved.

THEOREM 7.4. *RegTC captures* NLOGSPACE *on the class of linear constraint databases having the small coordinate property. Analogously, RegDTC captures* LOGSPACE.

8. CONCLUSION AND ONGOING WORK

We presented a family of query languages for linear constraint databases integrating recursion mechanisms into first-order logic. The fixed-points in these languages operate on a set of regions, which is essentially the set of faces in a decomposition of the input space. Note that the definition of the languages and their expressive power and complexity do not depend on a particular decomposition. Other decompositions could also be used, provided that they can be computed efficiently and it is possible to define a representation of the database from the decomposition.
Following this idea, it should also be possible to define interesting logics for other sorts of constraint databases, e.g. polynomial constraint databases.

It has been shown, that the logics defined capture all boolean queries decidable in important complexity classes. But the logics fail to capture, e.g., the class of non-boolean PTIME queries. We are currently working on extending the logics by a convex-closure operator such that the class of non-boolean PTIME-queries can be captured as well.

9. REFERENCES

[1] R. Anderson, P. Beame, and E. Brisson. Parallel algorithms for arrangements. *Algorithmica*, 15:104 – 125, 1996.

[2] F. Dumortier, M. Gyssens, L. Vanderurzen, and D. Van Gucht. On the decidability of semi-linearity for semi-algebraic sets and its implications for spatial databases. In *In Proc. 16th ACM Symp. on Principles of Database Systems*, 1997.

[3] H.-D. Ebbinghaus and J. Flum. *Finite Model Theory.* Springer, 1995.

[4] H. Edelsbrunner. *Algorithms in Combinatorial Geometry.* EATCS Monographs on Theoretical Computer Science. Springer, 1987.

[5] F. Geerts and B. Kuijpers. Expressing topological connectivity of spatial databases. DBPL'99, 1999.

[6] J. E. Goodman and J. O'Rourke, editors. *Handbook of Discrete and Computational Geometry.* CRC Press, 1997.

[7] M. T. Goodrich. Constructing arrangements optimal in-parallel. *Discrete & Computational Geometry*, 9:371 – 385, 1993.

[8] E. Grädel and Y. Gurevich. Metafinite model theory. *Information and Computation*, 140:26–81, 1998.

[9] E. Grädel and S. Kreutzer. Descriptive complexity theory for constraint databases. In *Computer Science Logic*, number 1683 in LNCS, pages 67 – 82. Springer, 1999.

[10] E. Grädel and K. Meer. Descriptive complexity theory over the real numbers. In *Mathematics of Numerical Analysis: Real Number Algorithms*, volume 32 of *AMS Lectures in Applied Mathematics*, pages 381–403. 1996.

[11] S. Grumbach and G. M. Kuper. Tractable recursion over geometric data. In *Principles and Practice of Constraint Programming*, number 1330 in LNCS, pages 450 – 462. Springer, 1997.

[12] S. Grumbach and J. Su. Finitely representable databases. *Journal of Computer and System Sciences*, 55:273–298, 1997.

[13] S. Grumbach and J. Su. Queries with arithmetical constraints. *Theoretical Computer Science*, 173:151–181, 1997.

[14] D. Harel. Towards a theory of recursive structures. In *Proceedings of 23rd International Symposium on Mathematical Foundations of Computer Science MFCS 98*, volume 1450 of *Lecture Notes in Computer Science*, pages 36–53. Springer, 1998.

[15] N. Immerman. *Descriptive complexity.* Graduate Texts in Computer Science. Springer, 1998.

[16] P. Kanellakis, G. Kuper, and P. Revesz. Constraint query languages. *Journal of Computer and Systems Sciences*, 51:26–52, 1995. (An extended abstract appeared in the Proceedings of PODS'90).

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

[18] B. Kuijpers, J. Paredaens, M. Smits, and J. Van den Bussche. Termination properties of spatial datalog programs. In *Logic in Databases*, number 1154 in LNCS, pages 101 – 116, 1996.

[19] G. Kuper, L. Libkin, and J. Paredaens, editors. *Constraint Databases.* Springer, 2000.

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

[21] L. Vanderurzen, M. Gyssens, and D. Van Gucht. On the desirability and limitations of linear spatial query languages. In *Proceedings of the 4th International Symposium of Spatial Databases*, number 951 in LNCS, pages 14 – 28, 1995.

[22] L. Vanderurzen, M. Gyssens, and D. Van Gucht. On query languages for linear queries definable with polynomial constraints. In *Proceedings of the 2nd International Conference on Principles and Practice of Constraint Programming*, number 1118 in LNCS, pages 468 – 481, 1996.

[23] G. M. Ziegler. *Lectures on Polytopes.* Number 152 in Graduate texts in mathematics. Springer, 1995.

APPENDIX

A. COMPUTING REGIONS IN NC[1]

In this section we precisely define the decomposition of databases into regions as sketched in Section 7.
Let $\mathfrak{B} := ((\mathbb{R}, <, +), S)$ be a linear constraint database, where S is a d-ary relation represented by $\varphi_S := \bigvee_i \psi_i$, $\psi_i := \bigwedge_j \varphi_{ij}$ and each φ_{ij} is a linear (in)equality. First, we separately define sets $\mathit{regions}(\psi_i)$ of regions for each disjunct ψ_i. The set of regions for S is defined as $\mathit{regions}(S) := \bigcup_i \mathit{regions}(\psi_i)$.
To ease notation we denote by ψ_i both the formula as well as the set of points it defines. It will always be clear from the context what is meant.
Let, for some i, $\psi := \psi_i$ be a disjunct of φ_S. We denote by $\mathit{closure}(\psi)$ the closure of the set of points defined by ψ. Let $\mathfrak{G}(\psi)$ be the set of (in)equalities occurring in ψ. Define

$$\mathfrak{H}(\psi) := \{\varphi \ : \ \begin{array}{l}\varphi \text{ is an equation contained in } \mathfrak{G}(\psi) \text{, or} \\ \text{there is an inequality } \varphi' \text{ occurring in } \psi \text{ and} \\ \varphi \text{ equals } \varphi' \text{ where the inequality is replaced} \\ \text{by equality}\}.\end{array}$$

as the set of formula defining the hyperplanes bounding ψ. We demonstrate the following algorithm using the examples given in Figure 9 and 10.

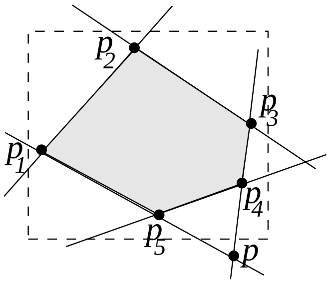

Figure 9: Example of a bounded polyhedron P.

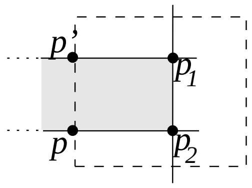

Figure 10: Example of an unbounded polyhedron P'.

First we calculate the set $vert(\psi)$ of vertices of ψ. For each d-tuple of atoms from $\mathfrak{H}(\psi)$ we compute the intersection of the hyperplanes. If they intersect in exactly one point p and p is contained in $closure(\psi)$, then p is a vertex.

Example. Consider the polytope P in Figure 9. The hyperplanes in $\mathfrak{H}(P)$ are the lines bounding the polytope. To compute the vertices every pair of lines is intersected. This results in the vertices $p_1, \ldots, p_5$ but also in points like p. Of course p is not a vertex of P as it is not contained in $closure(P)$.

The next step is to check whether ψ is bounded. Let c be the maximal absolute value of a coordinate of a vertex of ψ. If there are no vertices, then define $vert'(\psi)$ as the set of points p such that there is a d-tuple of atoms from $\mathfrak{H}(\psi) \cup \{x_i = 0 \ : \ 1 \leq i \leq d\}$ intersecting exactly in p. Let c be the maximal absolute value of a coordinate of a point in $vert'(\psi)$.

Define $cube(\psi)$ as $\{x_i = 2(c{+}1), x_i = -2(c{+}1) \ : \ 1 \leq i \leq d\}$. If each atom from $cube(\psi)$ has an empty intersection with ψ, then ψ is bounded, otherwise it is unbounded.

Example continued. In the example polyhedra, the $cube$'s are indicated by the dashed boxes. Obviously the (bounded) polytope P in Figure 9 does not intersect with the box, whereas the (unbounded) polyhedron does.

If ψ is bounded, then define the following regions for ψ. There are two sorts of regions, inner and outer regions. To define the inner regions, choose one vertex $p_{low} \in vert(\psi)$, e.g. the point whose coordinates are the lexicographically smallest. The set $region(\psi)$ consists of all open convex hulls C of $d{+}1$ vertices p_{low} and $p_1, \ldots, p_d$ of ψ, such that the line segment between p_{low} and any other vertex except $p_1, \ldots, p_d$ has an empty intersection with C.

Note that the points $p_{low}, p_1, \ldots, p_d$ do not have to be disjoint. Thus the regions may be of lower dimension than d.

The outer regions are defined as the open convex hull of at most d vertices $p_1, \ldots, p_d$, such that for no $1 \leq i, j \leq d$ the line segment between p_i and p_j intersects with the interior of ψ.

Example continued. Consider again the example polytope P. There are three 2-dimensional regions defined by $\{p_1, p_2, p_3\}$, $\{p_1, p_3, p_4\}$, and $\{p_1, p_4, p_5\}$. In addition there are seven 1-dimensional regions, namely the regions defined by $\{p_i, p_{i+1}\}$ for $1 \leq i \leq 4$, $\{p_1, p_5\}, \{p_1, p_3\}$, and $\{p_1, p_4\}$. Only the last two regions are inner regions, the other five are outer regions. Further, each point defines its own 0-dimensional (outer) region.

Now assume that ψ is unbounded. Define $icube(\psi) := \{x_i < 2(c+1), x_i > -2(c+1) \ : \ 1 \leq i \leq d\}$, where c is as above. As $icube(\psi)$ is bounded we can compute the vertices of $\psi \cap icube(\psi)$ as before. Define $up(\psi)$ as the set of all pairs $(p, (p-q))$ such that p is a vertex on the boundary of $icube(\psi)$, q is any other vertex, and the set $\{x \ : \ x = p + a(p-q)$ for some $a \geq 0\}$ is in $closure(\psi)$. Now define $regions(\psi)$ as follows. Compute the bounded regions for $vert(\psi \cap icube(\psi))$ as above. The unbounded regions are defined as follows.

Let $(p, p-q) \in up(\psi)$. Then $(p, p-q)$ defines the region $\{x \ : \ \text{there is an } a \in \mathbb{R}, a > 0 \text{ such that } x = p + a(p-q)\}$. The other unbounded regions are defined by the open convex hulls of at most d such regions.

Example continued. The vertices computed in the first step of the algorithm for the polyhedron P' are p_1 and p_2. The set defined by $icube$ is the interior of the dashed box and the vertices computed for $\psi \cap icube(\psi)$ are, besides p_1 and p_2, p and p'. Thus $regions(P')$ contains: two bounded 2-dimensional regions, $\{p, p', p_1\}$ and $\{p, p_1, p_2\}$, four bounded 1-dimensional regions, $\{p, p_2\}, \{p_2, p_1\}, \{p_1, p'\}$, and $\{p, p_1\}$, the last one being the only inner region, and four 0-dimensional regions. Further, there are the two unbounded 1-dimensional regions defined by $(p', p' - p_1)$ and $(p, p - p_2)$ and one unbounded 2-dimensional region.

Note that eventually the vertices of the cube can be contained in $vert(\psi \cap icube(\psi))$. This is necessary in certain degenerated cases, e.g. a polyhedron defined by only one inequality.

As mentioned above, the set $regions(\varphi_S)$ of regions of the input relation S is defined as the union of the regions for the disjuncts of φ_S. Note that every point $p \in S$ is contained in at least one region.

Although we do not precisely present the circuits, it should be clear that the computation of the regions essentially requires some applications of the Gauss and Fourier-Motzkin elimination method. Both methods can be computed by circuits with logarithmic depth and polynomial size, provided that the dimension, that is the number of variables involved, is fixed. See [20] for details. Thus we get the following lemma.

LEMMA A.1. *$regions(\varphi_S)$ can be computed by an NC^1 circuit.*