Query Languages for Constraint Databases: First-Order Logic, Fixed-Points, and Convex Hulls*

Stephan Kreutzer

Lehrgebiet Mathematische Grundlagen der Informatik,
RWTH Aachen, D-52056 Aachen,
kreutzer@informatik.rwth-aachen.de

Abstract. We define various extensions of first-order logic on linear as well as polynomial constraint databases. First, we extend first-order logic by a convex closure operator and show this logic, *FO(conv)*, to be closed and to have PTIME data-complexity. We also show that a weak form of multiplication is definable in this language and prove the equivalence between this language and the multiplication part of PFOL. We then extend *FO(conv)* by fixed-point operators to get a query languages expressive enough to capture PTIME. In the last part of the paper we lift the results to polynomial constraint databases.

1 Introduction

In recent years new application areas have reached the limits of the standard relational database model. Especially, geographical information systems, which are of growing importance, exceed the power of the relational model with their need to store geometrical figures, naturally viewed as infinite sets of points. Therefore, new database models have been proposed to handle these needs. One such data model is the framework of constraint databases introduced by Kanellakis, Kuper, and Revesz in 1990 [KKR90]. Essentially, constraint databases are relational databases capable of storing arbitrary elementary sets. These sets are not stored tuple-wise but by an elementary formula defining them. We give a precise definition of the constraint database model in Section 2. See [KLP00] for a detailed study of constraint databases.

When applied to spatial databases, one usually considers either databases storing semi-algebraic sets, called *polynomial constraint databases*, or semi-linear sets, known as *linear constraint databases*. Polynomial constraint databases allow the storage of spatial information in a natural way. The interest in the linear model results from the rather high (practical) complexity of query evaluation on polynomial databases. Essentially, the evaluation of first-order queries in the polynomial model consists of quantifier-elimination in the theory of ordered real fields, for which a non-deterministic exponential-time lower bound has

* Appeared in the Proc. of ICDT 2001, ©Springer-Verlag

been proven. On the other hand, query evaluation on linear constraint databases can be done efficiently. But first-order logic on these databases yields a query language with rather poor expressive power.

It is known that first-order logic lacks the power to define queries relying on recursion. A prominent example is connectivity which is not expressible in FO on almost all interesting classes of structures. This lack of expressive power shows up on polynomial as well as linear constraint databases. In finite model theory, a standard method to solve this problem is to consider least fixed-point logic, an extension of FO by a least fixed-point operator (see [EF95].) Although this logic has successfully been used in the context of dense-order constraint databases (see [KKR90,GS97,GK99]), it is not suitable for the linear database model, as it is neither closed nor decidable on this class of databases (see [KPSV96].) Logics extending FO by fixed-point constructs can be found in [GK97] and [GK00]. Decidability and closure of these languages was achieved by including some kind of stop condition for the fixed-point induction. In [Kre00], this problem has also been attacked resulting in a language extending FO by fixed-points over a finite set of regions in the input database.

Besides queries based on recursion there are also other important queries that are not first-order definable. In the context of linear databases a very important example of a query not expressible in FO is the convex closure query. We address this problem in Section 3. Unfortunately, any extension of FO by operators capable of defining convex hulls for arbitrary sets leads to a non-closed language. Therefore, we can only hope to extend FO consistently by a restricted convex hull operator. In this paper we add an operator to compute convex hulls of finite sets only. We show that this can be done in a consistent way, resulting in a language that is closed and has PTIME data-complexity. It is also shown that this language and PFOL, an extension of FO by restricted multiplication defined by Vandeurzen, Gyssens, and Van Gucht [VGG98], have the same expressive power.

As mentioned above, a language extending FO by recursion mechanisms has been defined in [Kre00]. Although this language turns out to be rather expressive for boolean queries, it lacks the power to define broad classes of non-boolean queries. In Section 4 we present two alternative approaches to define fixed-point logics on linear constraint databases. The first approach extends the logic defined in [Kre00] by the convex closure operator mentioned above, whereas the second approach combines convex closure with fixed-point induction over finite sets. It is shown that both approaches lead to query languages capturing PTIME on linear constraint databases.

In Section 5 we address the problem whether these results extend to the class of polynomial constraint databases. Clearly, extending first-order logic by a convex closure operator does not make sense, since convex closure is already definable in FO+POLY, that is, first-order logic on polynomial constraint databases. But it will be shown that the extension by fixed-point constructs can be suitably adapted to the polynomial setting, resulting in a language strictly more expressive than FO+POLY.

2 Preliminaries

Constraint databases. We first give a precise definition of the constraint database model. See [KLP00] for a detailed introduction. 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 $\varphi(x_1, \ldots, x_n)$ be a quantifier-free formula of vocabulary τ. We say that a n-ary relation $R \subseteq A^n$ is *represented by* $\varphi(x_1, \ldots, x_n)$ *over* $\mathfrak{A}$ iff R equals $\{\bar{a} \in A^n \ : \ \mathfrak{A} \models \varphi[\bar{a}]\}$. Let $\sigma := \{R_1, \ldots, R_k\}$ be a relational signature. A σ-constraint database over the context structure $\mathfrak{A}$ is a σ-expansion $\mathfrak{B} := (\mathfrak{A}, R_1, \ldots, R_k)$ of $\mathfrak{A}$ where all R_i are finitely represented by formulae φ_{R_i} over $\mathfrak{A}$. The set $\Phi := \{\varphi_{R_1}, \ldots, \varphi_{R_k}\}$ is called a *finite representation* of $\mathfrak{B}$.

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 depends on 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.

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 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 First-Order Logic and Convex Hulls

In this section we define an extension of first-order logic on semi-linear databases such that with each definable finite set of points also its convex hull becomes definable. Defining convex hulls is a very important concept when dealing with semi-linear databases but first-order logic itself is not powerful enough to define it. Thus, adding an operator allowing the definition of convex hulls results in a language strictly more expressive than FO. Note that allowing to define the convex closure of arbitrary point sets yields a non-closed query language, since multiplication - and thus sets which are not semi-linear - becomes definable. We therefore allow the definition of convex hulls for finite sets of points only.

Proviso. *In the rest of this paper, whenever we speak about the interior of a set or a set being open, we always mean "interior" or "open" with respect to the set's affine support.*

Definition 1 *Let $\bar{x}_i, \bar{x}'_i, \bar{y}$ denote sequences of l variables each and $\bar{z}$ denote a sequence of variables, such that all variables are distinct. The logic FO(conv) is defined as the extension of first-order logic by the following two rules:*

(i) If $\varphi \in$ FO(conv) is a formula with free variables $\{\bar{x}_1, \ldots, \bar{x}_k, \bar{z}\}$ then $\psi := [\mathrm{conv}_{\bar{x}_1,\ldots,\bar{x}_k}\varphi](\bar{y}, \bar{z})$ is also a formula, with free variables $\{\bar{y}, \bar{z}\}$.
(ii) If $\varphi \in$ FO(conv) is a formula with free variables $\{\bar{x}_1, \bar{x}'_1, \ldots, \bar{x}_k, \bar{x}'_k, \bar{z}\}$ then $\psi := [\mathrm{uconv}_{\bar{x}_1,\bar{x}'_1,\ldots,\bar{x}_k,\bar{x}'_k}\varphi](\bar{y}, \bar{z})$ is also a formula, with free variables $\{\bar{y}, \bar{z}\}$.

The semantics of the additional operators is defined as follows. Let $\mathfrak{B}$ be the input database and ψ and φ be as in Part (i) of the definition above. Let $\varphi^{\mathfrak{B}}$ be the result of evaluating the formula φ in $\mathfrak{B}$. If $\varphi^{\mathfrak{B}}$ is infinite, then $\psi^{\mathfrak{B}} := \varnothing$. Otherwise, $\psi^{\mathfrak{B}} := \{(\bar{a}, \bar{b}) \ : \ \bar{a} \in \bigcup \{\mathrm{conv}\{\bar{a}_1, \ldots, \bar{a}_k\} \ : \ \mathfrak{B} \models \varphi[\bar{a}_1, \ldots, \bar{a}_k, \bar{b}]\}\}$, where $\mathrm{conv}\{\bar{a}_1, \ldots, \bar{a}_k\}$ denotes the interior (with respect to the affine support) of the convex closure of $\{\bar{a}_1, \ldots, \bar{a}_k\}$.

The semantics of the uconv operator is defined similarly. The motivation for the uconv operator is, that every set defined by the conv operator is bounded. To overcome this restriction, the uconv operator is designed to handle "points at infinity". Let φ and ψ be as indicated in Part (ii) of the definition above and let $\mathfrak{B}$ be the input database. Again, if $\varphi^{\mathfrak{B}}$ is infinite, then $\psi^{\mathfrak{B}} := \varnothing$. Otherwise, $\varphi^{\mathfrak{B}} := \{(\bar{a}, \bar{b}) \ : \ $ there are $\bar{a}_1, \bar{a}'_1, \ldots, \bar{a}_k, \bar{a}'_k$ such that $\mathfrak{B} \models \varphi[\bar{a}_1, \bar{a}'_1, \ldots, \bar{a}_k, \bar{a}'_k, \bar{b}]$ and $\bar{a} \in \mathrm{conv}(\bigcup_{i=1}^{k} \mathit{line}(\bar{a}_i, \bar{a}'_i))\}$, where $\mathit{line}(\bar{a}_i, \bar{a}'_i) := \{\bar{x} \ : \ (\exists b \in \mathbb{R}^{\geq 0})$ such that $\bar{x} = \bar{a}_i + b(\bar{a}'_i - \bar{a}_i)\}$ defines the half line with origin $\bar{a}_i$ going through $\bar{a}'_i$.

Intuitively, each pair $(\bar{a}_i, \bar{a}'_i)$ represents two points, the point $\bar{a}_i$ and the point "reached" when starting at $\bar{a}_i$ and going in the direction of $\bar{a}'_i$ to infinite distance. Now uconv returns the union of the open convex closure (with respect to its affine support) of the $2k$ points represented by each tuple $((\bar{a}_{i,1}, \bar{a}'_{i,1}), \ldots, (\bar{a}_{i,k}, \bar{a}'_{i,k}))$.

Note that the condition on the formula φ to define a finite set is purely semantical. Since finiteness of a semi-linear set is first-order definable - consider, for example, the formula $\mathit{finite}(\varphi)$ stating that there are $\epsilon, \delta > 0$ such that the

Manhattan-distance between any two points in $\varphi^{\mathfrak{B}}$ is greater than ϵ and each point in $\varphi^{\mathfrak{B}}$ is contained in a hypercube of edge length δ - this condition can also be ensured on a syntactic level, thus giving the language an effective syntax.

We now give an example of a query definable in *FO(conv)*.

Example 2 *Let $\varphi(x)$ be a formula defining a finite set. The query*

$$mult_\varphi(x, y, z) := \{(a, b, c) \ : \ a \text{ satisfies } \varphi \text{ and } a \cdot b = c\}$$

is definable in FO(conv). We give a formula for the case $x, y, z \geq 0$. Let $\bar{x}_1 = (x_{1,1}, x_{1,2})$ and $\bar{x}_2 := (x_{2,1}, x_{2,2})$ be pairs of variables and $\psi(\bar{x}_1, \bar{x}_2; x) := \bar{x}_1 = (0,0) \wedge \bar{x}_2 = (1, x)$ be a formula defining for each x the two points $(0,0)$ and $(1, x)$ in $\mathbb{R}^2$. Then the formula

$$\psi(x, y, z) := [\mathrm{uconv}_{\bar{x}_1, \bar{x}_2} \varphi(x) \wedge \psi(\bar{x}_1, \bar{x}_2, x)](y, z; x)$$

defines the query $mult_\varphi$. The uconv operator defines - for the parameter x - the half line with origin $(0,0)$ and slope x. Thus the point (y, z) is on this line iff $x \cdot y = z$.

Theorem 3 *FO(conv) is closed and has* PTIME *data-complexity.*

The closure of the conv-operator follows from the finiteness of the sets, of which the convex closure is computed. PTIME data-complexity can easily be shown by induction on the structure of the queries.

We now compare the language to other extensions of FO for linear databases. Especially, we will show that *FO(conv)* and the language PFOL as defined by Vandeurzen et. al. [VGG98] have the same expressive power. We briefly recall the definition of PFOL (see [VGG98] and [Van99] for details.)

There are two different kinds of variables in PFOL, *real variables* and so-called *product variables*. A PFOL program consists of a sequence of formulae $(\varphi_1(x), \ldots, \varphi_k(x); \varphi(\bar{x}))$. Each φ_i is required to define a finite set $D_i \subset \mathbb{R}$. The formulae are allowed to use multiplication between variables $x \cdot p$, but at least p must be a product variable. All product variables must be bound at some point by a quantifier of the following form. In φ_i the quantifiers for the product variables are of the form $\exists p \in D_j$, where $j < i$. In φ all D_i may be used. Thus, essentially, the formulae φ_i define successively a sequence of finite sets and then at least one factor of each multiplication is restricted to one of these sets. In the original definition of PFOL there were also terms $t = \sqrt{|p|}$. We don't take this term building rule into account and allow only multiplication. We believe this to be the essential and important part of the language. But note that the square root operator strictly increases the expressive power of the language. Therefore we call the language considered here *restricted PFOL*.

It is known that convex closure can be defined in restricted PFOL. In Example 2 we already saw that multiplication with one factor bounded by a finite set can be defined in *FO(conv)*. Thus, the proof of the following theorem is straightforward.

Theorem 4 *Restricted PFOL = FO(conv).*

Note 5 *As the previous theorem shows, we can express atoms of the form $x \cdot_{\varphi} y = z$ in FO(conv), with the semantics being that $xy = z$ and $\varphi(x)$ holds, where φ is required to define a finite set. From now on, we allow atoms of this form in FO(conv)-formulae.*

The previous theorem implies that the bounds on expressiveness proven for PFOL carry over to *FO(conv)*. Here we mention one result which will be of special importance in Section 4.2.

It has been shown in [Van99] that there is a PFOL query which returns on a semi-linear set $S \subset \mathbb{R}^n$ a finite relation $S^{\mathrm{enc}} \subseteq \mathbb{R}^{n(n+1)}$ of $(n+1)$-tuples of points in $\mathbb{R}^n$, such that $S = \bigcup_{(\bar{a}_1,\ldots,\bar{a}_{n+1}) \in S^{\mathrm{enc}}} \mathrm{conv}(\bar{a}_1, \ldots, \bar{a}_{n+1})$. Thus, PFOL can compute a canonical finite representation of the input database and recover the original input from it. This will be used below to define a fixed-point query language capturing PTIME.

4 Query Languages Combining Convex Hulls and Recursion

In this section we present two approaches to combine the query language defined above with fixed-point constructs.

4.1 The Logic *RegLFP(conv)*

Let $S \subseteq \mathbb{R}^d$ be a semi-linear set. An arrangement of S is a partition of $\mathbb{R}^d$ into finitely many disjoint regions, i.e., connected subsets of $\mathbb{R}^d$, such that for each region R either $R \cap S = \varnothing$ or $R \subseteq S$. It is known that for a fixed dimension arrangements of semi-linear sets can be computed in polynomial time. See e.g. [Ede87] or [GO97] for details.

It follows from the definition that the input relation S can be written as a finite union of regions in its arrangement. In this section we consider a query language which has access to the set of regions in such an arrangement of the input database. This gives the logic access to the representation of the database increasing, thus, its expressive power. Precisely, we consider a fixed-point logic where the fixed-point induction is defined over the finite set of regions. The semantics of the logic is defined in terms of certain two-sorted structures, called *region extensions* of linear constraint databases. Let $\mathfrak{B} := ((\mathbb{R}, <, +), S)$ be a database and let $\mathcal{A}(S)$ be the set of regions in an arrangement of S. The logic then has separate variables and quantifiers for the reals and the set of regions.

We now give the precise definitions.

Definition 6 *Let $\mathfrak{B} := ((\mathbb{R}, <, +), S)$ be a linear constraint database, where S is a d-ary relation, and let $\mathcal{A}(S)$ be the set of regions of an arrangement of S. The structure $\mathfrak{B}$ gives rise to a two-sorted structure $\mathfrak{B}^{Reg} := ((\mathbb{R}, <, +), S; Reg, adj)$,* *called the* region extension of $\mathfrak{B}$*, with sorts* $\mathbb{R}$ *and* $Reg := \mathcal{A}(S)$ *and 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.*

The dimension of a region is defined as the dimension of its affine support, i.e., the dimension of the smallest affine subspace it is contained in. It is known that the number of regions in the arrangement is bounded polynomially in the size of the representation of S. As arrangements can be computed in polynomial time, one can compute the region extension of a given database in polynomial time as well.

We now define the logic *RegLFP(conv)* which is *FO(conv)* extended by a least fixed-point operator on the set of regions in the region extensions. In the definition of the logic we deal with three types of variables, so-called *element-*, *region-*, and *relation variables.* Element variables will be interpreted by real numbers, region variables by regions in the region extension of the database. Each relation variable is equipped with a pair $(k, l) \in \mathbb{N}^2$ of arities. Relation variables of arity (k, l) are interpreted by subsets $M \subseteq Reg^k \times \mathbb{R}^l$, such that for each $\bar{R} \in Reg^k$ the set $\{\bar{x} \ : \ (\bar{R}, \bar{x}) \in M\}$ is finitely representable.

Definition 7 *The logic RegLFP(conv) is defined as extension of FO(conv) on region extensions by a least fixed-point operator. Precisely, the logic extends FO(conv) by the following rules.*

- *If* R *is a region variable and* $\bar{x}$ *is a sequence of element variables, then* $R\bar{x}$ *is a formula.*
- *If* φ *is a formula and* R *a region variable, then* $\exists R\varphi$ *is also a formula.*
- *If* M *is a relation variable of arity* (k, l)*,* $\bar{R} := R_1, \ldots, R_k$ *is a sequence of region variables, and* $\bar{x} := x_1, \ldots, x_l$ *is a sequence of element variables, then* $M\bar{R}\bar{x}$ *is a formula.*
- *If* $\varphi(M, \bar{R}, \bar{x})$ *is a formula with free element variables* $\bar{x}$*, free region variables* $\bar{R} := R_1, \ldots, R_k$*, and a free* (k, l)*-ary relation variable* M*, such that* M *occurs only positively in* φ*, then* $[\mathrm{tLFP}_{M,\bar{R},\bar{x}}\varphi](\bar{R}, \bar{x})$ *is a formula.*
- *If* $\varphi(x)$ *is a formula and* x, y, z *are element variables, then* $x \cdot_\varphi y = z$ *is a formula.*

RegLFP(conv)-queries are defined by RegLFP(conv)-formulae without free region or relation variables.

The semantics of the new rules is defined as follows. An atom $R\bar{x}$ states that the point $\bar{x}$ is contained in the region R; an atom $M\bar{R}\bar{x}$ states that the tuple $(\bar{R}, \bar{x})$ is contained in M; a formula $\exists R\varphi$ states that there is a region $R \in Reg$ satisfying φ. The semantics of conv, uconv, and $\cdot_\varphi$ is defined as in the previous section.

Let M be a (k, l)-ary relation variable, $\bar{R} := R_1, \ldots, R_k$ be a sequence of region variables, $\bar{x} := x_1, \ldots, x_l$ be a sequence of element variables, and $\varphi(M, \bar{R}, \bar{x})$ be a formula positive in M. The result of the formula $\psi := [\mathrm{tLFP}_{M,\bar{R},\bar{x}}\varphi](\bar{R}, \bar{x})$

evaluated in a database $\mathfrak{B}$ is defined as the least fixed-point of the function f_φ defined as

$$\begin{aligned} f_\varphi : \mathrm{Pow}(\mathit{Reg}^k \times \mathbb{R}^l) &\longrightarrow \mathrm{Pow}(\mathit{Reg}^k \times \mathbb{R}^l) \\ M &\longmapsto \{(\bar{R}, \bar{a}) \in \mathit{Reg}^k \times \mathbb{R}^l \ : \ (\exists \bar{y}\,(\bar{R}, \bar{y}) \in M \wedge (\bar{R}, \bar{a}) \in M) \vee \\ &\qquad (\neg \exists \bar{y} (\bar{R}, \bar{y}) \in M \wedge \mathfrak{B} \models \varphi(M, \bar{R}, \bar{a}))\}. \end{aligned}$$

Clearly, the least fixed-point of the function exists and can be computed inductively in time polynomially in the number of regions and thus also in the size of the database.

Intuitively, we can think of the stages of the fixed-point induction as a set of tuples of regions $\overline{\mathrm{R}}$, where to each $\bar{R}$ there is a formula $\varphi_{\bar{R}}(\bar{x})$ attached to it defining a set of points in $\mathbb{R}^l$. But once a tuple of regions is contained in some stage of the fixed-point induction, the formula attached to it cannot be changed anymore. This is ensured by the first disjunct in the definition of f_φ. An example motivating this definition of the fixed-point operator is given below.

Note that there are different decompositions of $\mathbb{R}^d$ satisfying the conditions of an arrangement as presented above. Thus, the semantics of the logic depends on a particular decomposition chosen. But the results we prove below stay true for most decompositions, as long as the input can be recovered from them. Thus, for a given application area, one should choose a decomposition which is more intuitive to use.

Having defined the logic, we now give a motivating example for it.

Example 8 *Suppose we are given a road map with cities and the highways connecting them. Typically, the maps contain information about the distance between any two adjacent cities directly connected by a section of a highway. In this set-up a useful decomposition of the input into regions would be to have a region for each city, one for each section of a highway between two cities, as well as regions for the other parts of the map.*

Now suppose we want to travel from one city to another on a certain highway and we want to know the distance between these two cities. Let the constants s, t *denote the regions of the source and target city and assume a formula* $\mathrm{dist}(C, C', d)$, *stating that* C *and* C' *are regions of adjacent cities and* d *is the distance between both on the chosen highway. Then the formula*

$$\begin{aligned} \varphi(x) := [\mathrm{tLFP}_{M,C_1,C_2,d}\ & \mathrm{dist}(C_1, C_2, d) \ \vee \ (\exists C\, \exists d_1\, \exists d_2\ M(C_1, C, d_1) \wedge \\ & \mathrm{dist}(C, C_2, d_2) \ \wedge \ d = d_1 + d_2)\ ](s, t, x) \end{aligned}$$

defines the distance between the two cities. Here the real variable d *in the fixed-point induction is used to sum up the distances.*

We now show that the logic is expressive enough to capture all PTIME-queries on linear constraint databases.

Definition 9 *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* $\mathfrak{B}$.

Theorem 10 *RegLFP(conv) captures* PTIME *on the class of linear constraint databases having the small coordinate property.*

Sketch. As an arrangement of a semi-linear set can be computed in polynomial time, the region extension of the input database can be computed in PTIME as well. Now the PTIME data-complexity of *RegLFP(conv)* can be shown by induction on the structure of the queries. To prove that each PTIME-query can be defined by a *RegLFP(conv)*-query, we show that the run of a Turing-machine M computing the query can be simulated. The crucial point is that, given the input relation $S \subseteq \mathbb{R}^d$, a finite set $R \subseteq \mathbb{R}^{d(d+1)}$ of tuples of points such that $S = \bigcup\{\mathrm{conv}(\bar{a}_1, \ldots, \bar{a}_{d+1}) : (\bar{a}_1, \ldots, \bar{a}_{d+1}) \in R\}$ can be defined in *RegLFP(conv)*. Further, given such a set $R \subseteq \mathbb{R}^{l(l+1)}$ for some $l \in \mathbb{N}$, the set $\{\bar{x} : \bar{x} \in \mathrm{conv}(\bar{a}_1, \ldots, \bar{a}_{l+1})$ for some $(\bar{a}_1, \ldots, \bar{a}_{l+1}) \in R\}$ can be defined using the conv and uconv operators. This can be used to encode the input of a Turing-machine and to decode its output. The run of the Turing-machine can be simulated as usual in finite model theory, using region variables to denote positions on the Turing-tape. The restriction to databases with small coordinates comes from the fact, that in order to simulate the Turing-machine, coordinates of points have to be encoded by (tuples of) region variables. Thus, only polynomially many bits can be used to represent a coordinate of a point, restricting it to exponential size. $\square$

4.2 Finitary Fixed-Point Logic

The query language introduced in the previous section depends on a specific decomposition of the input database. Thus, its usability relies on the existence of a decomposition which can easily be understood by the user. Although this is the case in some application areas, it will be a problem in others. In this section we present a way to overcome this dependency on a specific, intuitive decomposition.

Below we define *finitary fixed-point logic* as the extension of *FO(conv)* by a least fixed-point operator over arbitrary definable finite sets. The idea is that, as mentioned in Section 3, the language *FO(conv)* is capable of defining a finite representation of the input database. Using this one can replace the fixed-point induction on the regions by a fixed-point induction on the finite representation.

We already mentioned that finiteness of a semi-linear set is first-order definable. Therefore we use formulae finite(φ), which, given a formula φ, evaluate to true if the set defined by φ is finite and false otherwise.

Definition 11 Finitary Fixed-Point Logic (FFP) *is defined as the extension of FO(conv) by a* finitary LFP operator. *Precisely, if $\bar{x} := x_1, \ldots, x_k$ and $\bar{z} := z_1, \ldots, z_l$ are sequences of first-order variables, R is a $(k+l)$-ary second-order variable, $\varphi(\bar{x})$ and $\psi(R, \bar{x}, \bar{z})$ are formulae, such that R occurs only positively in ψ and does not occur in φ, then also $[\mathrm{tLFP}_{R,\bar{x}}(\varphi, \psi)](\bar{u}, \bar{v})$ is a formula with free variables $\{\bar{u}, \bar{v}\}$, where $\bar{u}, \bar{v}$ are sequences of variables of arity k and l.*

The semantics of a formula $\chi := [\mathrm{tLFP}_{R,\bar{x}}(\varphi, \psi)](\bar{u}, \bar{v})$ is defined as the least fixed-point of the function

$$\begin{aligned} f_\chi : Pow(\mathbb{R}^{k+l}) &\longrightarrow Pow(\mathbb{R}^{k+l}) \\ R &\longmapsto \{(\bar{x}, \bar{z}) \ : \ (\exists \bar{y}\ (\bar{x}, \bar{y}) \in R \wedge (\bar{x}, \bar{z}) \in R) \vee \\ &\qquad ((\neg \exists \bar{y}\, (\bar{x}, \bar{y}) \in R) \wedge \mathfrak{B} \models \mathrm{finite}(\varphi) \wedge \varphi(\bar{x}) \wedge \psi(R, \bar{x}, \bar{z}))\}, \end{aligned}$$

where $\mathfrak{B}$ is the input database. Intuitively, the formula φ serves as a guard, ensuring that the fixed-point induction runs over the finite set defined by φ only. As before, the variables $\bar{z}$ can be used to attach some information to a tuple $\bar{x}$ contained in a induction stage.

Regarding the expressive power of this language, one can easily show that the languages *RegLFP(conv)* and FFP are equivalent. Thus, FFP captures PTIME on the class of linear constraint databases.

Theorem 12 *(i) FFP captures* PTIME *on the class of linear constraint databases having the small coordinate property.*
(ii) RegLFP(conv) and FFP have the same expressive power on the class of linear constraint databases.

5 Polynomial Constraint Databases

In this section we extend the approach taken in Section 4 to polynomial constraint databases. Clearly, it does not make sense to add a convex closure operator to first-order logic, as convex closure is already definable in FO on polynomial constraint databases. Thus, the logic *PolyLFP* is defined as the least fixed-point logic over region extensions, but the regions will now be defined by a cylindrical algebraic decomposition of the input space.

In Section 5.1 we give a (very brief) overview of cylindrical algebraic decompositions (CADs). See [Col75] or the monograph [CJ98] for details. We then define the region extension of polynomial constraint databases and introduce the logic *PolyLFP*.

The data-complexity and expressive power of the logic is considered thereafter. When dealing with complexity issues for constraint databases, one can base the examination on the Turing-model and restrict oneself to representing formulae with rational or real algebraic coefficients. Another approach is to ignore the complexity of the storage and manipulation of real numbers and use a computation model capable of storing arbitrary real numbers. These models have built-in functions for operations like multiplication and addition which can be executed in one time step. This approach puts more focus on the complexity of the underlying logic than on the complexity of manipulating numbers. Therefore we take this approach here and base our analysis of the complexity on the Blum-Shub-Smale (BSS) model. A brief introduction to BSS-machines can be found in Appendix A.

5.1 Cylindrical algebraic decomposition

Fix a dimension d. A *region* is defined as a connected subset of $\mathbb{R}^d$. The *cylinder* $Z(R)$ *over a region* R is defined as $R \times \mathbb{R}$. If R is a region and $f : R \to \mathbb{R}$ is a continuous function, then the *f-section of* $Z(R)$ is defined as the graph of f, that is, the set $\{(\bar{a}, b) \ : \ \bar{a} \in R \text{ and } b = f\bar{a}\}$.

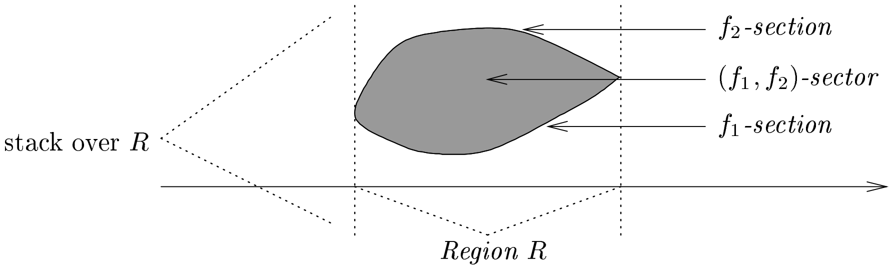

Fig. 1. Illustration of some concepts used in CADs.

Now, let $f_1, f_2 : R \to \mathbb{R}$ be continuous functions over R such that for all $\bar{a} \in R$ $f_1(\bar{a}) < f_2(\bar{a})$. The *$(f_1, f_2)$-sector of* $Z(R)$ is defined as the set $\{(\bar{a}, b) \ : \ \bar{a} \in R \text{ and } f_1(\bar{a}) < b < f_2(\bar{a})\}$. We allow f_1, f_2 to be the constant functions $-\infty, \infty$.

Let $X \subseteq \mathbb{R}^d$ be a set. A *decomposition* of X is a finite collection of disjoint regions whose union is X. Let R be a region and let $f_1, \ldots, f_k : R \to \mathbb{R}$ be a sequence of continuous functions from R to $\mathbb{R}$, such that $f_i(\bar{x}) < f_{i+1}(\bar{x})$ for all $\bar{x} \in R$. This sequence defines a decomposition of the cylinder $Z(R)$ into the regions defined by 1) the f_i-sections, 2) the (f_i, f_{i+1})-sectors where $i \in \{0, \ldots, k+1\}$ and $f_0(\bar{x}) = -\infty$ and $f_{k+1}(\bar{x}) = \infty$. Such a decomposition is called a *stack over R (determined by $f_1, \ldots, f_k$)*. See Figure 1 for an illustration of these concepts.

A decomposition D of $\mathbb{R}^d$ is called *cylindrical* if *i)* $d = 1$ and D is a stack over $\mathbb{R}^0$, i.e. a finite number of singletons and intervals, or *ii)* $d \geq 2$ and there is a cylindrical decomposition D' of $\mathbb{R}^{d-1}$ such that for each region $R \in D'$ there is a subset $S \subseteq D$ forming a stack over R. Clearly, a cylindrical decomposition of $\mathbb{R}^d$ determines a unique cylindrical decomposition of $\mathbb{R}^{d-1}$, called the *induced* decomposition.

A decomposition of $\mathbb{R}^d$ is called *algebraic*, if every region is a semi-algebraic set. A *cylindrical algebraic decomposition (CAD) of* $\mathbb{R}^d$ is a decomposition of $\mathbb{R}^d$ which is both cylindrical and algebraic.

Let $\mathcal{A} := \{f_1, \ldots, f_m\}$ be a set of polynomials from $\mathbb{R}^d$ to $\mathbb{R}$. A decomposition D of $\mathbb{R}^d$ is called *$\mathcal{A}$-invariant*, if for each $f_i \in \mathcal{A}$ and all regions $R \in D$, either for all $\bar{x} \in R$ $f_i(\bar{x}) > 0$, $f_i(\bar{x}) = 0$, or $f_i(\bar{x}) < 0$.

Let $\mathfrak{B} := (\mathbb{R}, <, +, \cdot, S)$ be a database, where S is d-ary. A CAD D of $\mathbb{R}^d$ is *invariant for* $\mathfrak{B}$, if it is invariant for the set of polynomials occurring in the representation of S. Thus, for each region $R \in D$ either $R \cap S = \varnothing$ or $R \subseteq S$.

It is known that a CAD of $\mathbb{R}^d$ invariant for a given set $\mathcal{A}$ of polynomials can be computed in double exponential time in the dimension d and polynomial time in the size of $\mathcal{A}$. Since the dimension is fixed, the algorithm operates in polynomial time in the size of $\mathcal{A}$, resp. in the size of a database $\mathfrak{B}$, if $\mathcal{A}$ is the set of polynomials occurring in the representation of $\mathfrak{B}$.

Further, a close analysis of the algorithms shows, that if D is a CAD of $\mathbb{R}^d$ invariant for a given set of polynomials of degree at most n, then, for d fixed, the degree of the polynomials defining the regions in D is polynomially bounded in n. See [Col75] for details.

5.2 A fixed-point logic for polynomial constraint databases

We define a fixed-point query language for polynomial constraint databases. In analogy to Section 4 this language is based on *region extensions* of databases.

Definition 13 *Let $\mathfrak{B} := ((\mathbb{R}, <, +, \cdot), S)$ be a polynomial constraint database. The* region extension $\mathfrak{B}^{Reg}$ of $\mathfrak{B}$ *is defined as a two-sorted structure $\mathfrak{B}^{Reg} := ((\mathbb{R}, <, +, \cdot), S; Reg)$. The first sort consists of the reals with order, addition, and multiplication, whereas the second sort consists of the set Reg of regions decomposing $\mathbb{R}^d$ in a CAD invariant for $\mathfrak{B}$.*

We now define the language *PolyLFP* as least fixed-point logic on region extensions. As before, the fixed-point induction is defined on the (finite) set of regions only. Since every database has a unique region extension we don't distinguish between databases and their region extensions and freely speak about a database being a model of a *PolyLFP* formula instead of explicitly mentioning its region extension.

Definition 14 *The logic PolyLFP is defined as the extension of first-order logic by the following rules. As in Definition 6 there are* element, region, *and* relation variables.

- *If R is a region variable and $\bar{x}$ is a sequence of element variables, then $R\bar{x}$ is a formula.*
- *If φ is a formula and R a region variable, then $\exists R\varphi$ is also a formula.*
- *If M is a relation variable of arity (k, l), $\bar{R} := R_1, \ldots, R_k$ are region variables, and $\bar{x} := x_1, \ldots, x_l$ are element variables, then $M\bar{R}\bar{x}$ is a formula.*
- *If $\varphi(M, \bar{R}, \bar{x})$ is a formula with free element variables $\bar{x}$, free region variables $\bar{R} := R_1, \ldots, R_k$, and a free (k, l)-ary relation variable M, such that φ is positive in M, then $[\mathrm{tLFP}_{M,\bar{R},\bar{x}}\varphi](\bar{R}, \bar{x})$ is a formula.*

PolyLFP-queries are defined by PolyLFP-formulae without free region or relation variables.

The semantics of the logic is defined analogously to *RegLFP(conv)*, with the region variables ranging over the set of regions in the region extension of a database and relation variables being interpreted as subsets of $Reg^k \times \mathbb{R}^l$. Since the region extension of a database can be computed in polynomial time and first-order logic on polynomial constraint databases has polynomial time data-complexity, the polynomial time data-complexity of *PolyLFP* follows immediately.

We now show that $\mathrm{P}_{\mathbb{R}}$ - the set of queries computable in polynomial time on a BSS-machine - can be captured by *PolyLFP* for a restricted class of polynomial constraint databases.

Definition 15 *A polynomial constraint database $\mathfrak{B}$ is said to be a k-degree database, if the highest degree of any variable of a polynomial in the representation of the database is at most k.*

Note that, as mentioned in Section 5.1, by bounding the degree of the polynomials in the input database, we also get a polynomial bound on the degree of the polynomials bounding the regions.

Theorem 16 *For each $k \in \mathbb{N}$, PolyLFP captures $\mathrm{P}_{\mathbb{R}}$ on the class of k-degree databases.*

Sketch. Again, the theorem is proved by showing that the run of a BSS-machine computing a $\mathrm{P}_{\mathbb{R}}$-query can be simulated, the crucial point being the representation of the input database. Recall from above, that the regions are either f-sectors or (f_1, f_2)-sections, for some polynomial functions f, f_1, f_2. We define a representation of the database by a disjunction of formulae defining the regions contained in it. Let region R be a f-section. Since, for some $k \in \mathbb{N}$, the input is restricted to be a k-degree database and thus the degree of f is bounded by k, the polynomial f can be defined as $f := \sum_{i \in \{0,\ldots,k\}^d} \bar{a}_i \bar{x}^i$, where, for $i := (i_0, \ldots, i_d) \in \{0, \ldots, k\}^d$, $\bar{x}^i$ denotes the product $\prod_{j=0}^{d} x_j^{i_j}$. The coefficients a_i in the sum can be defined by a formula $\varphi(\bar{a}) := \forall \bar{x} \exists y \, (\sum_{i \in \{0,\ldots,k\}^d} a_i \bar{x}^i = y \leftrightarrow R\bar{x}y)$, if the region R contains more than one point. The case where R contains only one point is trivial. The sectors can be defined similarly, since they are bounded by one or two sections. These sections are again regions and can thus be defined as described above. □

6 Conclusion

We introduced logics extending first-order logic by recursion mechanisms on linear as well as polynomial constraint databases. For linear constraint databases we also introduced a logic extending FO by the ability to compute convex closure and showed that regarding expressive power this concept equals the logic PFOL, where multiplication with one factor bounded by a finite set is permitted.

We showed that the fixed-point logics offer query languages with rather high expressive power, while still having tractable data-complexity. For practical implementations of these query languages, arrangements or cylindrical algebraic decompositions probably offer not the most intuitive definition of the region domain. But note that the results are independent of the precise definition of the region decomposition, as long as the database can be represented by a union of regions and the region decomposition can be computed in polynomial time. Thus, in a spatial database system, where spatial information is combined with non-spatial information giving a meaning to part of the spatial image, one could use a decomposition consistent with this semantical part.

References

[BCSS98] L. Blum, F. Cucker, M. Shub, and S. Smale. *Complexity and Real Computation.* Springer, 1998.

[BSS89] L. Blum, M. Shub, and S.Smale. On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. *Bulletin of the American Mathmatical Society*, 21:1–46, 1989.

[CJ98] B.F. Caviness and J.R.Johnson, editors. *Quantifier Elimination and Cylindric Algebraic Decomposition.* Springer, 1998.

[Col75] George E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Automata Theory and Formal Languages*, number 33 in LNCS, pages 134–183, Berlin, 1975. Springer-Verlag.

[Ede87] H. Edelsbrunner. *Algorithms in Combinatorial Geometry.* EATCS Monographs on Theoretical Computer Science. Springer, 1987.

[EF95] H.-D. Ebbinghaus and J. Flum. *Finite Model Theory.* Springer, 1995.

[GK97] 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.

[GK99] E. Grädel and S. Kreutzer. Descriptive complexity theory for constraint databases. In *Computer Science Logic*, number 1683 in LNCS, pages 67 – 82. Springer, 1999.

[GK00] F. Geerts and B. Kuijpers. Linear approximation of planar spatial databases using transitive-closure logic. In *PODS 2000*, pages 126–135. ACM Press, 2000.

[GO97] Jacob E. Goodman and Joseph O'Rourke, editors. *Handbook of Discrete and Computational Geometry.* CRC Press, 1997.

[GS97] S. Grumbach and J. Su. Finitely representable databases. *Journal of Computer and System Sciences*, 55:273–298, 1997.

[KKR90] P. C. Kanellakis, G. M. Kuper, and P. Z. Revesz. Constraint query languages. In *PODS 1990*, pages 299–313, 1990.

[KLP00] G. Kuper, L. Libkin, and J. Paredaens, editors. *Constraint Databases.* Springer, 2000.

[KPSV96] 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.

[Kre00] S. Kreutzer. Fixed-point query languages for linear constraint databases. In *PODS 2000*, pages 116–125. ACM press, 2000.

[Van99] L. Vandeurzen. *Logic-Based Query Languages for the Linear Constraint Database Model*. PhD thesis, Limburgs Universitair Centrum, 1999.

[VGG98] L. Vandeurzen, M. Gyssens, and D. Van Gucht. An expressive language for linear spatial database queries. In *PODS 1998*, pages 109–118, 1998.

A Blum-Shub-Smale-Machines

Blum-Shub-Smale (BSS) machines have been introduced by Blum, Shub, and Smale in 1989 [BSS89]. We give a brief review of the computation model here. For a detailed introduction see [BSS89] and the monograph on real computation [BCSS98]. Note that we use a slightly different presentation of BSS machines than the presentation given by Blum, Shub, and Smale.

Intuitively, a BSS machine is a random access machine with real registers and built-in operations for addition, subtraction, multiplication, and division. A precise definition is given below.

Definition 17 *We define $\mathbb{R}^\infty$ as the set of all infinite sequences $(a_0, a_1, \dots)$ of real numbers such that there is $k \in \mathbb{N}$ with $a_i = 0$ for all $i \geq k$.*

Definition 18 *A BSS machine consists of the input space $\mathbb{R}^\infty$, the state space $S := \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathbb{R}^\infty$, the output space $\mathbb{R}^\infty$ and a directed connected graph $G := (\{0, \dots, N\}, V)$ for some $N \in \mathbb{N}$. Each vertex has at most two children and one of the following five types of operations assigned to it. The graph represents the program of the machine, where vertices can be thought of as commands and successors of nodes as the possible successive commands.*

- *The node 0 is the* input node. *An input $(y_1, y_2, \dots, y_k, 0, \dots) \in \mathbb{R}^\infty$ is mapped to $(1, 1, 1, y_1, y_2, \dots) \in S$. Thus the node 0 has the successor 1.*
- *N is the* output node. *The machine stops in the position $(N, i, j, x_1, x_2, \dots)$ and outputs $(x_1, x_2, \dots)$. The node N has no successors.*
- Computation nodes: *An operation of this type transforms a state $(n, i, j, x_1, x_2, \dots) \in S$ to $(n+1, i', j', g_n(x_1, x_2, \dots)) \in S$, where $n+1$ is the successive command, $i' \in \{i+1, 1\}, j' \in \{j+1, 1\}$, and g_n is one of the following basic operations:*
  - *Two registers x_r, x_s are added, subtracted, multiplied, or divided and the result is stored in x_r.*
  - *A register x_r is set to a real constant.*

  The registers x_i with $i \notin \{r, s\}$ are not altered.
- Test nodes: *A test node n has exactly two successors $n+1$ and $\beta(n)$. On a state $(n, i, j, x_1, \dots) \in S$ the machine tests whether $x_1 \geq 0$ and continues in node $n+1$ if the answer is* yes *and in node $\beta(n)$ otherwise.*
- Copy nodes: *In state $(n, i, j, x_1, \dots)$, the machine sets $x_j := x_i$ and continues at node $n+1$.*

Based on BSS machines one can define complexity classes like $\mathrm{P}_\mathbb{R}$ as the class of all problems computable on a BSS machine in polynomially many steps. See [BCSS98] for details.