An Expressive Language for Linear Spatial Database Queries

(extended abstract)

Luc Vandeurzen*
University of Limburg
lvdeurze@luc.ac.be

Marc Gyssens*
University of Limburg
gyssens@charlie.luc.ac.be

Dirk Van Gucht†
Indiana University
vgucht@cs.indiana.edu

Abstract

We exhibit a coordinate-based language, called PFOL, which is sound for the linear queries computable in first-order logic over the reals and extends the latter's restriction to linear arithmetic. To evaluate its expressive power, we first consider PFOL-fin, the PFOL queries that compute finite outputs upon finite inputs. In order to study this fragment of PFOL, we also define a syntactical language, called SPFOL, which is safe with respect to queries from finite inputs to finite outputs. We show that SPFOL has the same expressive power as SafeEuQl [15], whence all ruler-and-compass constructions in the plane on finite sets of points can be expressed in SPFOL. This result gives a geometrical justification of SPFOL, and highlights the richness of PFOL-fin. Then, we define finite representations for arbitrary semi-linear sets and show that there are PFOL programs for both the encoding and the decoding. This result is used (*i*) to identify a broad, natural class of linear queries expressible in PFOL, highlighting the richness of general PFOL, and (*ii*) to establish a general theorem about lifting query languages on finite databases to query languages on arbitrary linear databases. This theorem is applied to a recent result of Benedikt and Libkin [5] from finite to arbitrary semi-linear sets, yielding the existence of a natural, syntactically definable fragment of FO + poly sound and complete for all FO + poly-expressible linear queries.

1 Introduction

Following the seminal work by Kanellakis, Kuper, and Revesz [13] on constraint query languages with polynomial constraints, various researchers have introduced geometric database models and query languages within this framework [12, 16]. We adopt the formalism of [16], which we shall call the polynomial spatial database model, in which both geometric objects and queries are expressed using polynomial inequalities. Geometric objects described by polynomial inequalities are called semi-algebraic sets, and the query language using polynomial inequalities is referred to as FO + poly.

Several authors [1, 2, 7, 11, 10, 12, 13, 18, 19] discussed linear spatial database models which can be seen as linear restrictions of the polynomial database model. These linear models suffice for the majority of applications encountered in GIS, geometric modeling, and spatial and temporal databases [14, 17]. Geometric objects described by linear inequalities are called semi-linear sets, and the restriction of FO + poly using only linear inequalities is referred to as FO + linear.

Unfortunately, not all linear queries (i.e., mappings between spatial databases describable in the linear model) expressible in FO + poly can be described in FO + linear [2]. The present authors showed that, in fact, a whole class of rather straightforward linear queries, among which computing convex hull and deciding colinearity, are not expressible in FO + linear [19]. Several attempts have been made to enrich the expressive power of FO + linear, but this turns out to be a difficult task. Afrati et al. [2] and the present authors [18] showed that naive extensions of FO + linear easily cease to remain sound with respect to the linear queries, and yield a language equivalent in expressive power to FO + poly. Sound ways to extend FO + linear have been proposed (e.g., [8, 19]), but the geometric significance of these extensions is not always clear.

This objection also holds for FO + linear itself, which is a second reason why FO + linear is not such a desirable linear query language: unlike for FO + poly, no geometric characterization of FO + linear is known. This observation has lead some authors to consider point-based linear languages [15].

In this paper, we try to obtain a geometrically meaningful, but still coordinate-based, extension of FO + linear. We relax the condition that FO + linear formulae may only contain linear inequalities by introducing a second sort of variables, the so-called *product variables*. These product variables must be quantified and are allowed to range over a finite set of numbers only. In formulae, product variables may occur in multiplications with other product variables or with ordinary real variables. The language thus obtained will be called PFOL.

To justify that PFOL is geometrically meaningful, we first consider PFOL-fin, the PFOL queries that compute finite outputs upon finite inputs. Since PFOL-fin is semantically defined, we also propose a syntactical language, called SPFOL, which is safe with respect to queries from finite inputs to finite outputs. We show that SPFOL has the same

*Dept. WNI, University of Limburg, Universitaire Campus, B-3590 Diepenbeek, Belgium. Contact author is Luc Vandeurzen.
†Computer Science Dept., Indiana University, Bloomington, IN 47405-4101, USA.

PODS '98 Seattle WA USA
Copyright ACM 1998 0-89791-996-3/98/6...$5.00

expressive power as SafeEuQl [15], a language which captures the notion of ruler-and-compass constructions on finite sets of points in the plane. This result not only gives a geometrical justification of SPFOL, but also highlights the richness of PFOL-fin.

To lift our results to more general linear queries, we propose a technique to encode an arbitrary semi-linear set into a finite database, called its *finite representation*, as well as a decoding technique to restore the original semi-linear sets. We show that both encoding and decoding can be expressed in PFOL. This result yields a "lifting theorem" which guarantees that a sensible query language on arbitrary semi-linear databases can be obtained by defining a sensible query language from finite inputs to finite outputs, and letting this language operate on finite representations.

The results described in the previous paragraph are used in two ways in this paper.

First, we argue that natural linear queries are (*i*) FO + poly-expressible and (*ii*) satisfy the condition that the finite representation of the output can somehow be "constructed" from the finite representation of the input. We formalize this intuition by defining a linear query to be *constructible* if the corresponding query between the finite representations is FO+poly-expressible, and if a finite superset of the active domain of the finite representation of the output can be computed from the finite representation of the input in SPFOL (or, equivalently, in SafeEuQl). To some extent, the second condition for constructibility can be seen as an extension of the notion of domain preservation in the relational model to the linear spatial database model. The constructible linear queries are a vast class of FO + poly-expressible linear queries; in particular, they include all Boolean queries, as well as, for example, the convex-hull query. We show that all constructible queries can be expressed in PFOL.

Second, we apply the "lifting theorem" to recent results of Benedikt and Libkin [5], which the authors kindly communicated to us. Benedikt and Libkin exhibit a syntactically definable fragment of FO + poly which is sound and complete for the FO+poly-expressible queries from finite inputs to finite outputs, which they claim to be natural. From this, it follows there exists a query language which is sound and complete for the FO + poly-expressible linear queries, and which is perhaps more natural than the one proposed in [9]. It should be noted that the latter languages was proposed precisely to encourage the search for more natural query languages which are sound and complete for the FO + poly-expressible linear queries.

We conclude by pointing out some directions for future research.

2 Preliminaries

2.1 The polynomial and linear spatial database models

We first review the *polynomial model* [13, 16]. The polynomial model is described using the first-order language of the ordered field of the real numbers $(\mathbf{R}, \leq, +, \times, 0, 1)$, i.e., the language $(\leq, +, \times, 0, 1)$. Every real formula $\varphi(x_1, \ldots, x_n)$ with free real variables among $x_1, \ldots, x_n$ defines a geometrical figure $\{(x_1, \ldots, x_n) \mid \varphi(x_1, \ldots, x_n)\}$ in n-dimensional Euclidean space $\mathbf{R}^n$. Point-sets defined in this way are called *semi-algebraic sets*.

A spatial *database scheme* $\mathcal{S}$ is a finite set of *relation names*. Each relation name R has a fixed arity associated to it, corresponding with the dimension of the Euclidean space containing the semi-algebraic sets that will be the instantiation of R.[1] A database scheme has type $[n_1, \ldots, n_k]$ if the scheme consists of relation names, say $R_1, \ldots, R_k$, respectively of arity $n_1, \ldots, n_k$.

A *syntactic* spatial *database instance* is a mapping $\mathcal{I}$ assigning to each relation name R of a database scheme $\mathcal{S}$ a syntactic spatial relation $\mathcal{I}(R)$ of the same type. A *syntactic* spatial *relation* of arity n is a real formula $\varphi(x_1, \ldots, x_n)$, with n free variables. The semantics of a syntactic database instance $\mathcal{I}$ over a database scheme $\mathcal{S}$ is the mapping I assigning to each relation name R in $\mathcal{S}$ the semantic spatial relation $I(\mathcal{I}(R))$, which is the semi-algebraic set defined by the syntactic spatial relation $\mathcal{I}(R)$.

In the polynomial model, we consider a query of signature $[n_1, \ldots, n_k] \to [n]$ to be a mapping from instances of a spatial database scheme of type $[n_1, \ldots, n_k]$ to instances of a spatial database scheme of type $[n]$ that can be regarded in a consistent way both at the syntactic and semantic level, and is computable at the syntactic level.

In this context, we define FO + poly as the query language obtained by adding to the language of real formulae atomic formulae of the form $R(x_1, \ldots, x_n)$, with $x_1, \ldots, x_n$ real variables and R a relation name of arity n. A query of signature $[n_1, \ldots, n_k] \to [n]$ is definable in FO + poly if there exists an FO + poly formula φ with n free real variables such that, for every input database instance of signature $[n_1, \ldots, n_k]$, $\{(x_1, \ldots, x_n) \mid \varphi(x_1, \ldots, x_n)\}$ evaluates to the corresponding output database, which is of type $[n]$.

From the polynomial model, a linear spatial database model can be obtained by only considering real formulae containing linear polynomials. The corresponding restrictions of semi-algebraic sets, spatial databases, queries, and the language FO + poly are called *semi-linear sets*, linear spatial databases, linear queries, and FO + linear, respectively. We allow the coefficients of the linear polynomials involved to be arbitrary real algebraic numbers (the maximalistic approach in [9]). In this way, all semi-algebraic polytopes[2] are semi-linear.

Queries of signature $[n_1, \ldots, n_k] \to [0]$ are called *Boolean* queries, because the sets $\{()\}$ and $\{\}$ can be seen as encoding the truth values *true* and *false*, respectively. Since both these sets are semi-linear, every Boolean query induces a linear query, when restricted to linear inputs.

We conclude this overview by mentioning that linear topological queries, such as taking the closure, the interior, or the boundary, are expressible in FO + linear [18].

Where necessary, or where useful for the purpose of abbreviation, we use vector notation to denote points. Equalities or inequalities between vectors must be interpreted coordinate-wise.

2.2 Structural properties of semi-linear sets

We also need some notions and results introduced and discussed in earlier work of the present authors with Dumortier [9], which we recall here in a more informal style.

Given a semi-linear set S of $\mathbf{R}^n$, we say that a point $\vec{p}$ of S is a *regular point* if there exists a sufficiently small neighborhood V of $\vec{p}$ such that $S \cap V$ and the affine support of $S \cap V$ coincide within V. If, moreover, $S \cap V$ has the same dimension as S, then $\vec{p}$ is called a *regular point of maximal dimension* of S.

[1] In general, spatial databases can also contain a non-spatial part. In this paper, we are not concerned with this non-spatial part, and, therefore, do not include it to simplify the presentation.

[2] A polytope is the convex closure of a finite set of points.

Example 2.1 Consider the semi-linear set

$$S = \{(x,y,z) \mid (0 \leq x \leq 3 \wedge 0 \leq y \leq 3 \wedge 0 \leq z \leq 3) \vee (3 \leq x \leq 5 \wedge y = 1 \wedge z = 0) \vee (x = 5 \wedge y = 5 \wedge z = 0)\}$$

in three-dimensional space, shown in Figure 1, which consists of a closed filled cube with a closed line segment attached to it at the point $(3,1,0)$ and an isolated point.

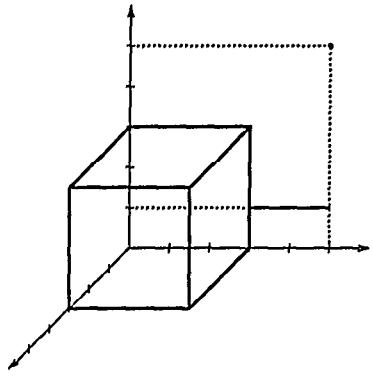

Figure 1: The semi-linear set of Example 2.1.

The regular points of S are the points of the open cube, the points of the open line segment, and the isolated point. The regular points of maximal dimension of S are only the points of the open cube. □

We recall that the set of the regular points of maximal dimension $Reg(S)$ of a semi-linear set S is again semi-linear, and that the corresponding linear query can be expressed in FO + linear.

From a semi-linear set S, several *special point-sets* can be computed, as follows. If S is not a discrete set of points, we compute $Reg(S)$. Then, we consider $S - Reg(S)$, the complement of $Reg(S)$ with respect to S, and $\overline{S} - Reg(S)$, the complement of $Reg(S)$ with respect to $\overline{S}$, the topological closure of S. Notice that both sets, which can be computed in FO + linear, have strictly lower dimension than S. We repeat the produce on both sets independently. In this way, we obtain zero-, one-, two-, three-,... up to at most n-dimensional point-sets. We are particularly interested in the zero-dimensional point-sets, which are called *special points*.

By construction, the linear query computing the set of special points of a semi-linear set is expressible in FO + linear.

Example 2.2 Consider again the semi-linear set S of Example 2.1, displayed in Figure 1. Then $S - Reg(S)$ and $\overline{S} - Reg(S)$ coincide and consist of the faces of the cube, the closed line segment attached to it, and the isolated point, which constitute a two-dimensional semi-linear set, say T. We see that $Reg(T)$ consists of the open faces of the cube. Again, $T - Reg(T)$ and $\overline{T} - Reg(T)$ coincide. They consist of the edges of the cube, the closed line segment attached to it, and the isolated point, which constitute a one-dimensional semi-linear set, say U. We see that $Reg(U)$ consists of the open edges of the cube, and the open line segment attached to it. Once more, $U - Reg(T)$ and $\overline{U} - Reg(T)$ coincide. They consist of the corner points of the cube, the end points of the line segment attached to it, and the isolated point, which constitutes a discrete set of points. This set of 11 points is the set of special points of S. □

It must be noted that, in general, the special points of a semi-linear set S need not belong to S (although, of course, they always belong to the topological closure $\overline{S}$ of S). Also, the sets $S - Reg(S)$ and $\overline{S} - Reg(S)$ do *not* always coincide, as was the case in Example 2.2. In general, *both* sets must be considered to ensure that no special points are missed.

To highlight the significance of the special points, we need to make the following observation. It has been shown [9] that, if the affine supports of all the special point-sets of a semi-linear set, including the special points themselves, are described as intersections of $(n-1)$-dimensional hyperplanes[3], then S is a finite union of cells of the partition of n-dimensional space induced by all these hyperplanes.

In the case of a *bounded* semi-linear set S, all these special point-sets (whence their affine supports) are supported by special points. Hence, if we restrict the partition meant in the previous paragraph to the affine support of S, say A, we can obtain this partition by considering hyperplanes of the affine space A which are supported by special points. We illustrate this on an example.

Example 2.3 Consider the semi-linear set

$$S = \{(x,y) \mid (3x - y > 2 \wedge y > 1 \wedge x + 2y < 10) \wedge \neg(x \leq 3 \wedge y \geq 2 \wedge x \geq y) \wedge \neg(1 \leq y \leq 3 \wedge x = 3)\}$$

in the plane, shown in Figure 2, which consists of an open triangle, out of which a closed triangle and a closed line segment have been cut.

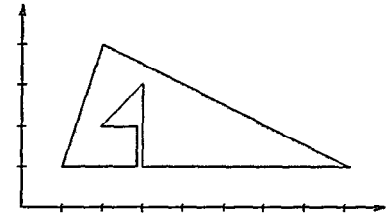

Figure 2: The semi-linear set of Example 2.3.

It can easily be seen that there are 7 special points, namely the corner points of the two triangles involved, and the missing point on the base of the outer triangle. Without elaborating, we mention that there are 8 one-dimensional special point-sets, namely the open line segments in which the special points divide the boundary of S. The affine support of S is the plane itself, so hyperplanes are lines. Hence, to obtain the partition meant above, we must certainly consider the 7 lines supporting the 8 special line segments. Since these lines can also be used to determine the special points, they suffice. Figure 3 shows these 7 lines (the dots indicating the boundary of S). Coordinate axes are hidden for clarity.

These lines, together with the open half-planes they define, induce a partition of the entire plain, consisting of 12 points, 31 open line segments or half-lines, and 20 open regions. In Figure 3, these regions have been identified by numbers. Notice that S is indeed the finite union of cells of this partition, namely the open regions 13, 14, 15, 17, 18, and 19, all of which are filled polygons, and the open intervals separating them. □

[3]Mathematically, n-dimensional space itself can be obtained as the intersection of the empty family of $(n-1)$-dimensional hyperplanes.

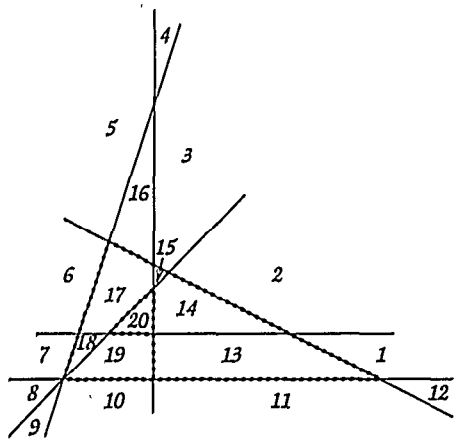

Figure 3: The lines needed to describe the affine supports of the special point-sets of the semi-linear set Example 2.3, shown in Figure 2. The numbers indicate the open regions in the induced partition.

An *unbounded* semi-linear set S in n-dimensional space is still a finite union of cells of the partition of n-dimensional space induced by any finite collection of $(n-1)$-dimensional hyperplanes for which all special point-sets of S can be described as an intersection of some of those hyperplanes. Unfortunately, such a collection of hyperplanes can in general no longer be obtained from the special points of S alone.

Example 2.4 Consider the semi-linear set $S = \{(x,y) \mid 0 \leq y \leq 2x\}$ in the two-dimensional plane, shown as the closed angular sector in Figure 4 (coordinate axes are hidden for clarity).

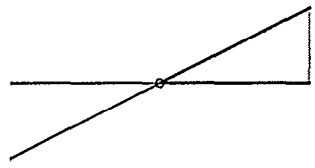

Figure 4: The semi-linear set of Example 2.4.

The one-dimensional special point-sets of S are the two open half-lines on the boundary of S. The only special point of S is the angular point, which is marked in Figure 4. The partition induced by the affine supports of the two open half-lines (these lines form the required set of hyperplanes in the plane) consists of 4 open angular sectors (one of which is S), 4 open half lines, and the angular point. Obviously, the 4 lines defining this partition cannot be obtained from the angular point alone. □

To solve the problem sketched above, the concept of *bounding box* was introduced in [9]. A bounding box of a semi-linear set S in n-dimensional space is an open hypercube C (for practical purpose usually centered around the origin $\vec{0}$ of R^n), of which the length of the edges is a real algebraic number, such that each special point-set of S has a non-empty intersection with C. In particular, a bounding box of a semi-linear set contains all its special points. In the full version of [9], we exhibit an FO+linear-expressible linear query which returns a bounding box for a given semi-linear set. Given a semi-linear set S, this query first computes the special points of each of the 2^n intersections of S with one of the hyperquadrants of R^n, and then finds an open hypercube centered around the origin of the coordinate system containing all these points. Once a bounding box C is found for a semi-linear set S of R^n, the desired partition can be obtained from the special points of $S \cap C$, as this latter semi-linear set is bounded.

Example 2.5 Consider again the semi-linear set S of Example 2.4, displayed in Figure 4. Figure 5 shows a bounding box C for S.

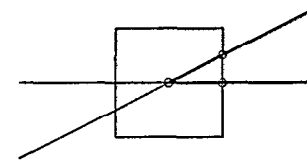

Figure 5: A bounding box for the semi-linear set of Example 2.4, shown in Figure 2.4.

Notice that $S \cap C$ is a triangle, and that the special points of $S \cap C$ are the corner points of this triangle, which are marked in Figure 5. Notice that the two lines defining the partition of the two-dimensional plane described in Example 2.4 are supported by the special points of $S \cap C$. □

3 The languages PFOL, PFOL-fin, and SPFOL

First, we present PFOL, the query language around which this paper revolves. The main idea behind the introduction of this language is to augment FO + linear with a limited amount of multiplicative power, as is needed, for instance, to express colinearity or to compute convex hull or distances, without sacrificing the soundness of the language with respect to the class of linear queries. We show that PFOL is indeed sound with respect to the class of FO+poly-expressible linear queries, and illustrate its expressive power with some examples.

Second, in order get to get a better understanding of the properties of PFOL, we consider two restricted versions of PFOL in which queries return finite outputs upon finite inputs. We define PFOL-fin to consist of the PFOL queries that compute finite outputs upon finite inputs. Since PFOL-fin is semantically defined, we also propose a syntactical language, called SPFOL, which is safe with respect to queries from finite inputs to finite outputs.

3.1 The language PFOL

In order to define PFOL queries, we first need to define FO + linear + $\mathrm{P}(D_1,\ldots,D_k)$ formulae, where $D_1,\ldots,D_k$ are so-called domain symbols denoting finite sets. For this purpose, we assume two sorts of variables, called *real variables* and *product variables*. We shall use $x,y,z,\ldots$, possibly subscripted, to denote real variables, and $p,q,r,\ldots$, possibly subscripted, to denote product variables. Finally, we shall use the symbol t, possibly subscripted, to denote a variable that can either be a real variable or a product variable.

Definition 3.1 Let $D_1,\ldots,D_k$ be domain symbols. An FO + linear + $\mathrm{P}(D_1,\ldots,D_k)$ formula is built using the connectives $\neg$ and $\wedge$ and the quantifiers $(\exists x)$, with x a real variable, and $(\exists p \in D_i)$, with p a product variable and $1 \leq i \leq k$, from the following atomic formulae:

- $R(t_1,\ldots,t_n)$, with R a predicate symbol of type $[n]$ and $t_1,\ldots,t_n$ real variables or product variables;
- $\sum_{i=1}^{n} a_i t_i \ \theta \ a$ with $a_1,\ldots,a_n$, and a real algebraic numbers, $t_1,\ldots,t_n$ real variables or product variables, and $\theta \in \{=,\neq,<,\leq,>,\geq\}$;
- $t_1 = pt_2$, with t_1 and t_2 real variables or product variables and p a product variable; and
- $t = \sqrt{p}$ with t a real variable or a product variable and p a product variable.

Furthermore, in an FO + linear + $\mathrm{P}(D_1,\ldots,D_k)$ formula, each product variable must be bound by an appropriate quantifier.

In the context of sets of real numbers as interpretations for $D_1,\ldots,D_k$ and a linear spatial database containing interpretations for the relevant predicate symbols, the semantics of an FO+linear+$\mathrm{P}(D_1,\ldots,D_k)$ formula is the obvious one. □

Notice that, for $k = 0$, the above definition reduces to the definition of an FO + linear formula.

We now state an important soundness property of an FO + linear + $\mathrm{P}(D_1,\ldots,D_k)$ formula.

Proposition 3.2 *Let $D_1,\ldots,D_k$ be domain symbols and let $\varphi(x_1,\ldots,x_n)$ be an* FO + linear + $\mathrm{P}(D_1,\ldots,D_k)$ *formula with free variables $x_1,\ldots,x_n$. If $D_1,\ldots,D_k$ are interpreted as* finite *sets of real algebraic numbers, then $\{(x_1,\ldots,x_n) \mid \varphi(x_1,\ldots,x_n)\}$ is interpreted as a semi-linear set.*

Proof. Let, for $i = 1,\ldots,k$, the interpretation of D_i be the finite set of real algebraic numbers $\{c_{i1},\ldots,c_{im_i}\}$. We consecutively eliminate product variables in φ by replacing formulae of the form $(\exists p \in D_i)\psi$ by $\psi(c_{i1}/p)\vee\ldots\vee\psi(c_{im_i}/p)$ (or *false*, if D_i is interpreted as the empty set). Hence, if $\overline{\varphi}$, is the result of all the substitutions, then $\{(x_1,\ldots,x_n) \mid \varphi(x_1,\ldots,x_n)\} = \{(x_1,\ldots,x_n) \mid \overline{\varphi}(x_1,\ldots,x_n)\}$. Since the latter set is expressed as the result of an FO + linear query on a linear spatial database, it is semi-linear. □

Using Definition 3.1, we now define the language PFOL.

Definition 3.3 A PFOL program is of the form

$$D_1 \leftarrow \varphi_1(x);\ldots;D_k \leftarrow \varphi_k(x);\ \{(x_1,\ldots,x_n) \mid \varphi(x_1,\ldots,x_n)\},$$

with $D_1,\ldots,D_k$ domain symbols, for $i = 1,\ldots,k$, φ_i an FO + linear + $\mathrm{P}(D_1,\ldots,D_{i-1})$ formula, and $\varphi(x_1,\ldots,x_n)$ an FO + linear + $\mathrm{P}(D_1,\ldots,D_k)$ formula.

The semantics of a PFOL program is as follows. First, $D_1,\ldots,D_k$ are consecutively interpreted as finite sets of real algebraic numbers. In this process, D_i is interpreted as the set $\{x \mid \varphi_i(x)\}$ if this set is finite, and as the empty set otherwise.[4,5] Next, the set $\{(x_1,\ldots,x_n) \mid \varphi(x_1,\ldots,x_n)\}$ is interpreted in the obvious way. □

From Proposition 3.2, the following is immediate.

Theorem 3.4 *The language PFOL is sound with respect to the* FO + poly*-expressible linear queries.*

In other words, the limited multiplicative power that has been added to FO + linear did not destroy the soundness of the language with respect to linearity. To illustrate the gain in expressive power entailed by this limited addition of multiplicative power, we give examples of PFOL programs for queries known to be inexpressible in FO + linear [19].

In order to write down PFOL programs concisely, we first remark that the syntax of an FO + linear + $\mathrm{P}(D_1,\ldots,D_k)$ formula may be relaxed without increasing the expressive power of PFOL. In particular, on can allow atomic formulae of the form

$$\sum_{i=1}^{n} P_i(p_1,\ldots,p_m)t_i \ \theta \ a,$$

with $P_1,\ldots,P_n$ polynomials with real algebraic coefficients over m variables, $p_1,\ldots,p_m$ product variables, $t_1,\ldots,t_n$ real variables or product variables, and a a real algebraic number. Similarly, we may allow atomic formulae of the form

$$t = \sqrt{P(p_1,\ldots,p_m)},$$

with t a real variable or a product variable, P a polynomial with real algebraic coefficients, and $p_1,\ldots,p_m$ product variables. Finally, we can also allow disjunction and subformula of the form $(\forall p \in D)\varphi$ as abbreviation of $\neg(\exists p \in D)\neg\varphi$.

In the following examples, the binary predicate symbol R represents a semi-linear set in the two-dimensional plane.

Example 3.5 The PFOL program

$$\begin{array}{l} D_1 \leftarrow (\exists y)(R(x,y) \vee R(y,x)); \\ \{() \mid (\forall p \in D_1)(p \neq p) \ \vee \\ \quad (\exists p_x \in D_1)(\exists p_y \in D_1)(\exists q_x \in D_1)(\exists q_y \in D_1) \\ \quad (\forall r_x \in D_1)(\forall r_y \in D_1)(R(r_x,r_y) \Rightarrow \\ \quad (\exists \lambda)(r_x = p_x\lambda + q_x - q_x\lambda \wedge r_y = p_y\lambda + q_y - q_y\lambda))\} \end{array}$$

decides whether the points stored in R are colinear, if R is finite, and returns *false* otherwise. □

Example 3.6 The PFOL program

$$\begin{array}{l} D_1 \leftarrow (\exists y)(R(x,y) \vee R(y,x)); \\ \{(x,y) \mid (\exists p_x \in D_1)(\exists p_y \in D_1)(\exists q_x \in D_1)(\exists q_y \in D_1) \\ \quad (\exists r_x \in D_1)(\exists r_y \in D_1)(\exists \lambda_1)(\exists \lambda_2)(\exists \lambda_3) \\ \quad (R(p_x,p_y) \wedge R(q_x,q_y) \wedge R(r_x,r_y) \wedge \\ \quad \lambda_1 \geq 0 \wedge \lambda_2 \geq 0 \wedge \lambda_3 \geq 0 \wedge \\ \quad \lambda_1 + \lambda_2 + \lambda_3 = 1 \wedge x = p_x\lambda_1 + q_x\lambda_2 + r_x\lambda_3 \wedge \\ \quad y = p_y\lambda_1 + q_y\lambda_2 + r_y\lambda_3)\} \end{array}$$

computes the convex hull of the points stored in R, if R is finite, and the empty set otherwise.

The modification of the above PFOL program obtained by removing the conditions $\lambda_1 \geq 0$, $\lambda_2 \geq 0$, and $\lambda_3 \geq 0$ computes the affine support of the points stored in R, if R is finite, and the empty set otherwise. □

Example 3.7 The PFOL program

$$\begin{array}{l} D_1 \leftarrow (\exists y)(R(x,y) \vee R(y,x)); \\ \{(x) \mid (\exists p_x \in D_1)(\exists p_y \in D_1)(\exists q_x \in D_1)(\exists q_y \in D_1) \\ \quad (R(p_x,p_y) \wedge R(q_x,q_y) \ \wedge \\ \quad x = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2})\} \end{array}$$

computes the set of all distances between pairs of points stored in R, if R is finite, and the empty set otherwise. □

[4] For a semi-linear set S, finiteness is equivalent to discreteness [6], which can be decided in FO + linear by the formula $(\forall \vec{x})(\exists \vec{\varepsilon})(\vec{\varepsilon} \neq \vec{0} \wedge S(\vec{x}) \Rightarrow \neg(\exists \vec{y})(S(\vec{y}) \wedge \neg(\vec{x} = \vec{y}) \wedge \vec{x} - \vec{\varepsilon} < \vec{y} < \vec{x} + \vec{\varepsilon}))$.

[5] By Proposition 3.2, the set $\{x \mid \varphi_i(x)\}$ is semi-linear. If, in addition, it is finite, it must therefore consist of real algebraic numbers.

One can also write PFOL programs deciding whether the points of an *arbitrary* semi-linear set of R^n are colinear, or computing the convex hull or the affine support of such a set. However, we shall not exhibit such programs, as their existence—as well as the way in which they can be obtained—will follow from Theorem 6.4 and its proof, further on in this paper.

3.2 The language PFOL-fin

Definition 3.8 A PFOL-fin program is a PFOL program which returns finite outputs upon finite inputs. □

Example 3.9 The PFOL programs in Examples 3.5 and 3.7 are examples of PFOL-fin programs. □

3.3 The language SPFOL

Clearly, the language PFOL-fin is defined semantically. Therefore, we define syntactically a version of PFOL, called SPFOL, which is "safe" with respect to finite databases.

Definition 3.10 Let $D_1, \ldots, D_k$ be domain symbols. An atomic partial safe FO + linear + $\mathrm{P}(D_1, \ldots, D_k)$ formula is one of the following:

- *true*;
- $adom(t)$, with t a real variable or a product variable;
- $R(t_1, \ldots, t_n)$, with R a predicate symbol of type $[n]$ and $t_1, \ldots, t_n$ real variables or product variables;
- $Q_1(p_1, \ldots, p_m)x = Q_2(p_1, \ldots, p_m)$, with $p_1, \ldots, p_m$ product variables, x a real variable, and Q_1 and Q_2 expressions built from the product variables $p_1, \ldots, p_m$ and the rational numbers, using addition, multiplication, and square rooting; or
- $Q(p_1, \ldots, p_m)\ \theta\ 0$, with $p_1, \ldots, p_m$ product variables and Q an expression built from the product variables $p_1, \ldots, p_m$ and the rational numbers, using addition, multiplication, and square rooting.

A partial safe FO + linear + $\mathrm{P}(D_1, \ldots, D_k)$ formula is one of the following:

- an atomic partial safe FO + linear + $\mathrm{P}(D_1, \ldots, D_k)$ formula;
- $\varphi(x_1, \ldots, x_n, p_1, \ldots, p_m) \vee \psi(x_1, \ldots, x_n, p_1, \ldots, p_m)$, with φ and ψ partial safe FO + linear + $\mathrm{P}(D_1, \ldots, D_k)$ formulae with the same free variables;
- $\varphi(x_1, \ldots, x_n, p_1, \ldots, p_m) \wedge \psi(y_1, \ldots, y_r, q_1, \ldots, q_l)$, with φ and ψ partial safe FO + linear + $\mathrm{P}(D_1, \ldots, D_k)$ formulae;
- $\varphi(x_1, \ldots, x_n, p_1, \ldots, p_m) \wedge \neg\psi(y_1, \ldots, y_r, q_1, \ldots, q_l)$, with φ and ψ partial safe FO + linear + $\mathrm{P}(D_1, \ldots, D_k)$ formulae, $\{y_1, \ldots, y_r\} \subseteq \{x_1, \ldots, x_n\}$, and $\{q_1, \ldots, q_l\} \subseteq \{p_1, \ldots, p_m\}$; or
- $(\exists p_i \in D_j)\varphi(x_1, \ldots, x_n, p_1, \ldots, p_m)$, with φ a partial safe FO + linear + $\mathrm{P}(D_1, \ldots, D_k)$ formula, $1 \leq i \leq m$, and $1 \leq j \leq k$.

A safe FO + linear + $\mathrm{P}(D_1, \ldots, D_k)$ formula is a partial safe FO + linear + $\mathrm{P}(D_1, \ldots, D_k)$ formula in which each product variable is bound by an existential quantifier. In the context of sets of real numbers as interpretations for $D_1, \ldots, D_k$ and a linear spatial database containing interpretations for the relevant predicate symbols, the semantics of a safe FO + linear + $\mathrm{P}(D_1, \ldots, D_k)$ formula is the obvious one, provided *adom* is interpreted as the active domain of the input database.

The syntax and semantics of an SPFOL program are the same as the syntax and semantics of an arbitrary PFOL program, except that only safe formulae are allowed. □

Example 3.11 The PFOL program in Example 3.7 is actually an SPFOL program, provided D is defined using the active-domain atom. □

The observations in Example 3.11 generalize as follows:

Proposition 3.12 *1. Linear queries expressible in SPFOL are also expressible in PFOL.*

2. SPFOL queries return finite outputs upon finite inputs.

The first statement is obvious, whereas the second statement can be proved by structural induction, but is also a corollary to Theorem 4.3 in the following section.

It remains an open question whether *all* PFOL-fin queries can be expressed in SPFOL. However, it can be deduced from a result of Benedikt and Libkin [3, 4] that all *Boolean* PFOL-fin queries can be expressed in SPFOL. The reader is invited to translate the PFOL program in Example 3.5 into an SPFOL program.

4 Expressiveness of SPFOL

Although SPFOL is a coordinate-based language, we claim that it is nevertheless geometrically meaningful. To substantiate this claim, we prove that SPFOL has the same expressive power as SafeEuQl [15] for queries in the two-dimensional plane. Indeed, SafeEuQl has been devised to capture precisely the ruler-and-compass constructions on finite sets of points in the two-dimension plane which return finite sets of points. We take the liberty of redefining SafeEuQl here, so that the definition may better accommodate proofs by structural induction.

We assume the existence of constants $\vec{0}$, $\vec{e}_1$, and $\vec{e}_2$, which represent the origin of the canonical coordinate system and both unit coordinate vectors. Variables, denoted as $\vec{p}, \vec{q}, \vec{r}, \ldots$, and possibly subscripted, represent points rather than coordinates. Predicates, denoted as $\hat{R}(\vec{p}_1, \ldots, \vec{p}_m)$, and possibly subscripted, represent relations consisting of m-ary tuples of points.

Definition 4.1 The SafeEuQl formulae are the smallest collection of formulae closed under the following rules:

- *true* is a safe formula;
- $\hat{R}(\vec{p}_1, \ldots, \vec{p}_m)$ is a safe formula;
- if $\varphi(\vec{p}_1, \ldots, \vec{p}_m)$ is a safe formula, and if $\{\vec{q}_1, \ldots, \vec{q}_6\} \subseteq \{\vec{p}_1, \ldots, \vec{p}_m\}$, and if $\vec{r}$ is an arbitrary variable, then
  - $\varphi(\vec{p}_1, \ldots, \vec{p}_m) \wedge \vec{r} = \vec{0}$,
  - $\varphi(\vec{p}_1, \ldots, \vec{p}_m) \wedge \vec{r} = \vec{e}_1$,
  - $\varphi(\vec{p}_1, \ldots, \vec{p}_m) \wedge \vec{r} = \vec{e}_2$,
  - $\varphi(\vec{p}_1, \ldots, \vec{p}_m) \wedge \vec{r} = \vec{q}_1$,
  - $\varphi(\vec{p}_1, \ldots, \vec{p}_m) \wedge \textbf{spec-pred}(\vec{r}, \vec{q}_1, \ldots, \vec{q}_6)$,

  where spec-pred stands for one of the *special predicates* considered in [15], are safe formulae;

- if $\varphi(\vec{p}_1,\ldots,\vec{p}_m)$ and $\psi(\vec{p}_1,\ldots,\vec{p}_m)$ are safe formulae, then $\varphi(\vec{p}_1,\ldots,\vec{p}_m) \vee \psi(\vec{p}_1,\ldots,\vec{p}_m)$ is a safe formula;
- if $\varphi(\vec{p}_1,\ldots,\vec{p}_m)$ and $\psi(\vec{q}_1,\ldots,\vec{q}_k)$ are safe formulae, then $\varphi(\vec{p}_1,\ldots,\vec{p}_m) \wedge \psi(\vec{q}_1,\ldots,\vec{q}_k)$ is a safe formula;
- if $\varphi(\vec{p}_1,\ldots,\vec{p}_m)$ and $\psi(\vec{q}_1,\ldots,\vec{q}_k)$ are safe formulae for which $\{\vec{q}_1,\ldots,\vec{q}_k\} \subseteq \{\vec{p}_1,\ldots,\vec{p}_m\}$, then $\varphi(\vec{p}_1,\ldots,\vec{p}_m) \wedge \neg\psi(\vec{q}_1,\ldots,\vec{q}_m)$ is a safe formula; and
- if $\varphi(\vec{p}_1,\ldots,\vec{p}_m)$ is a safe formula, and $1 \leq i \leq m$, then $(\exists\vec{p}_i)\varphi(\vec{p}_1,\ldots,\vec{p}_m)$ is a safe formula.

The semantics of a SafeEuQl formulae is the obvious one, provided the special predicates are interpreted as in [15].

A SafeEuQl query is of the form

$$\{(\vec{p}_1,\ldots,\vec{p}_m) \mid \varphi(\vec{p}_1,\ldots,\vec{p}_m)\},$$

and has the usual semantics. □

Example 4.2 To illustrate Definition 4.1, we give an example of a SafeEuQl query involving the special predicate l-l-crossing$(\vec{r},\vec{q}_1,\ldots,\vec{q}_4)$,[6] which evaluates to *true* if $\vec{r}$ is the intersection point of the lines defined by $\vec{q}_1$ and $\vec{q}_2$, respectively, $\vec{q}_2$ and $\vec{q}_3$.[7]

Let R be a finite set of points in the two-dimensional plane. The SafeEuQl query

$$\{(\vec{x}) \mid (\exists\vec{p})(\exists\vec{q})(\exists\vec{r})(\exists\vec{s})(R(\vec{p}) \wedge R(\vec{q}) \wedge R(\vec{r}) \wedge R(\vec{s}) \wedge \text{l-l-crossing}(\vec{x},\vec{p},\vec{q},\vec{r},\vec{s})\}$$

is a SafeEuQl query computing the intersection points of all pairs of different non-parallel lines supported by points of R. □

For a comparison between SPFOL with SafeEuQl to make sense, we must realize that SPFOL works with variables representing real numbers, whereas SafeEuQl works with variables representing points in the two-dimensional plane. We shall, therefore, only consider SPFOL programs in which all relation predicates involved, as well as the output, have even arity. This assumption allows us to interpret consecutive variables in a relational predicate, or in the output format, as coordinates of a point in the two-dimensional plane. Taking this observation into account, we now come to the main result of this section:

Theorem 4.3 *The languages SPFOL and SafeEuQl are equivalent in expressive power.*

Proof. (Sketch.) *From SafeEuQl to SPFOL*: First, we observe that for each free point variable in a SafeEuQl formula, only a finite number of points can be substituted to satisfy the formula. Hence, all point variables in a SafeEuQl query can be translated into pairs of product variables, provided that, upon completing the translation of the SafeEuQl formula into an SPFOL program, conjuncts of the form $x = p$, with x a real variable and p a product variable, are added to produce the output. The actual translation goes by structural induction, in which both the domains needed to quantify product variables and the subformulae under consideration are translated simultaneously. The basis of the induction is the active domain for the computation of the domains, and the obvious translation of relation predicates for the formulae.

From SPFOL to SafeEuQl: To simplify the translation, we can again bring free variables to the outer level of the formulae involved by adding conjuncts of the form $x = p$, with x a real variable and p a product variable. The translation requires a double, nested structural induction: at the outer level, the induction goes on the structure of the SPFOL program, and, at the inner level, the induction goes on the structure of the constituting safe formulae. To do the translation, we observe that, in the SafeEuQl query to be constructed, the point variables that occur in input relational predicates correspond to pairs of product variables in the input relational predicates of the given SPFOL program. However, it is much easier to represent a product variable p by a point variable $\vec{p}$ corresponding to the point $(p,0)$. This transition is possible in SafeEuQl, as the points $\vec{p} = (p,0)$ and $\vec{q} = (q,0)$ can be constructed from the point $\vec{r} = (p,q)$ with ruler and compass. This technique can be used to compute the representation of the active domain from the input relations. To obtain the output point variables of the SafeEuQl program, which correspond to pairs of output variables of the SPFOL program, the reverse construction is required, which can also be done with ruler and compass. For the actual translation, we observe that there are ruler-and-compass constructions for addition, subtraction, multiplication, division, and square rooting. To make sure that, during the computation, point variables are appropriately bound to input relations through successive constructions, we can add, for each range-restricted existential quantifier, a conjunct stating that the quantified variable belongs to the domain under consideration. □

[6] Dummy variables have been omitted.

[7] If $\vec{q}_1$ equals $\vec{q}_2$, or $\vec{q}_3$ equals $\vec{q}_4$, then the predicate evaluates to *false*.

5 Finite representations of arbitrary semi-linear sets

A way to lift query languages defined on finite databases to query languages defined on more general databases is by exhibiting a finite representation for the latter, together with encoding and decoding algorithms. This idea has also been considered by Kuijpers et al. [15], in the context of SafeEuQl, and by Benedikt and Libkin, on a more abstract level.

In this paper, we exhibit a finite representation for arbitrary linear databases, together with encoding and decoding algorithms that are expressible in PFOL. The techniques used are derived from the properties of semi-linear sets described in Section 2.2. Rather than defining the notion of finite representation first, we define it as the result of the encoding algorithm. For clarity, we first consider the case where the linear database consists of *bounded* semi-linear sets, and then say which modifications are required if unbounded semi-linear sets occur. We also discuss the ramifications of our result.

5.1 The bounded case

Let S be a bounded semi-linear set of $\mathbf{R}^n$. We describe the encoding. First, we compute the set of special points T of S, which can be done in FO+linear [9]. Next, we determine the affine support of T, which can be done in PFOL (Example 3.6). We also determine the dimension of this affine support, say $k \leq n$, which can be done in FO+linear [19]. Then, consider all possible sequences $\vec{p}_{11},\ldots,\vec{p}_{1k},\ldots,\vec{p}_{k1},\ldots,\vec{p}_{kk}$ of k^2 points of T. Let, for $i = 1,\ldots,k$, A_i be the affine support of $\vec{p}_{i1},\ldots,\vec{p}_{ik}$. Let U consists of all points $\vec{p}$ for which $\cap_{i=1}^{k} A_i = \{\vec{p}\}$. Clearly, $T \subseteq U$, and U can be computed from T in PFOL. (Where necessary, product variables can be introduced which range over the set of coordinates of points in T.)

Example 5.1 We recall that, if S is the two-dimensional semi-linear set shown in Figure 2, then the set T of the special points of S consists of the three corner points of the outer triangle, the three corner points of the inner triangle, and the special point on the base of the outer triangle, 7 points in total. All 6 lines shown in Figure 3 connect points of T S, so the 12 points they generate (including the 7 points of T) all belong to U. The set U, however, contains many more points. For instance, two additional points are obtained by intersecting the line connecting the tops of both triangles with the two horizontal lines in Figure 3. □

Next, we compute finite relations $R_1, \ldots, R_{n+1}$, where, for $i = 1, \ldots, n+1$, R_i has arity $i \times n$. The relation R_1 consists of all points of U contained in S; the relation R_2 consists of all pairs of different points of U such that the open interval between those points is contained in S; the relation R_3 consists of all triples of non-colinear points of U such that the open triangle of which these points are the corners is contained in S; ...; in general, the relation R_i consists of all i-tuples of non-co-$(i-1)$-dimensional points of U such that the open[8] convex hull of these points is contained in S. Notice that, necessarily, $R_{k+2}, \ldots, R_{n+1}$ are empty. Since product variables can be used to represent the points of the finite relation U, and since the convex hull of a finite set of points can be computed in PFOL (Example 3.6), we conclude that the entire query of type $[n] \to [n, 2n, \ldots, (n+1)n]$ which, given a semi-linear set S as input, returns the linear spatial database consisting of the finite relations $R_1, \ldots, R_{n+1}$ as output, can be computed in PFOL. The output database is the *finite representation* of S.

We consider the following property to be the key property of a finite representation:

Proposition 5.2 *Let S be a bounded semi-linear set and let $R_1, \ldots, R_{n+1}$ be a finite representation of S. Then*

$$S = \bigcup_{i=1}^{n+1} \bigcup_{(\vec{p}_1, \ldots, \vec{p}_i) \in R_i} \mathrm{CH}(\vec{p}_1, \ldots, \vec{p}_i),$$

where $\mathrm{CH}(\vec{p}_1, \ldots, \vec{p}_i)$ is the the open convex hull of $\vec{p}_1, \ldots, \vec{p}_i$.

Proof. (Sketch.) The inclusion from right to left is trivial; therefore, we concentrate on the other inclusion. From Section 2.2, we know that S is a finite union of cells of the partition of the affine support A of S induced by the $(k-1)$-dimensional hyperplanes supported by the points of T. By construction, the cells of this partition have their corner points in U. Since S is bounded, S is a finite union of bounded cells, and since these are convex, they can be triangulated using their corner points only. Thus, each cell contained in S, whence S itself, is fully contained in the left-hand side of the above equality. □

Decoding the finite representation of a bounded semi-linear set can be done in PFOL in the obvious way, suggested by the formula in the statement of Proposition 5.2.

5.2 The general case

Let S be a semi-linear set of $\mathbf{R}^n$ which may be unbounded. If a partition of $\mathbf{R}^n$ such that S is a finite union of cells must be obtained from special points, it is necessary to introduce a bounding box C, which can be computed in FO+linear, as explained at the end of Section 2.2. The methodology of the bounded case can easily be extended, provided we can incorporate into the finite representation the "corner points at infinity" of the unbounded cells of the partition. "Points at infinity" will be represented by directional vectors. Hence, for each direction, we consider two "points at infinity," one for each orientation.

Thus, we construct in PFOL from $S \cap C$ the set T of special points and the set U of "finite" corner points, in the same way as in the bounded case. To find the "corner points at infinity," we compute the set V of directional vectors $\vec{s} = \pm(\vec{c} - \vec{p})$, where $\vec{c}$ is a point of T on the boundary of C and $\vec{p}$ is an arbitrary point of T, and the open interval between both points is fully contained in S. Using the coordinates of the points of T as a finite domain for restricting the range of product variables, we can easily write a PFOL program to compute V.

The *finite representation* of S contains again finite relations $R_1, \ldots, R_{n+1}$, but, now, for $i = 1, \ldots, n+1$, R_i has arity $i \times (n+1)$. Intuitively, we add an extra coordinate to each point, which equals "1" for a "finite" point, and "0" for a "point at infinity," i.e., for a directional vector.

For $i = 1, \ldots, n+1$, R_i is the subset of

$$\bigcup_{j=1}^{i} (U \times \{1\})^j \times (V \times \{0\})^{i-j}$$

consisting of all i-tuples

$$((\vec{p}_1, 1), \ldots, (\vec{p}_j, 1), (\vec{s}_{j+1}, 0), \ldots, (\vec{s}_i, 0))$$

such that the "finite" and "infinite" points involved are not co-$(i-1)$-dimensional and such that the open convex hull of all these points is contained in S. We notice that a point p is in this open convex hull if there exist $\lambda_1 > 0, \ldots, \lambda_j > 0, \mu_{j+1} \geq 0, \ldots, \mu_i \geq 0$ such that $\lambda_1 + \cdots + \lambda_j = 1$ and

$$\vec{p} = \lambda_1 \vec{p}_1 + \cdots + \lambda_j \vec{p}_j + \mu_{j+1} \vec{s}_{j+1} + \cdots + \mu_i \vec{s}_i.$$

Since product variables can be used to represent the points of the finite relations U and V, we conclude that the entire query of type $[n] \to [n+1, 2(n+1), \ldots, (n+1)^2]$, that, given a semi-linear set S as input, returns its finite representation as output, can be computed in PFOL.

Example 5.3 Consider again the two-dimensional set S of Example 2.4, shown with a bounding box C in Figure 5. For convenience, we shall assume that the angular point has coordinates $(0,0)$, while the special points on the bounding box have coordinates $(1,0)$ and $(1,0.5)$, respectively. The set T consists of those points. Clearly, $U = T$. The set V consists of 6 directional vectors, with coordinates $(1,0)$, $(-1,0)$, $(1,0.5)$, $(-1,-0.5)$, $(0,0.5)$, and $(0,-0.5)$, respectively. The finite representation of S is as follows,

$$\begin{aligned}
R_1 &= \{(0,0,1), (1,0,1), (1,0.5,1)\};\\
R_2 &= \{(0,0,1;1,0,1), (0,0,1;1,0.5,1), (1,0,1;1,0.5,1),\\
&\quad (0,0,1;1,0,0), (0,0,1;1,0.5,0), (1,0,1;1,0,0),\\
&\quad (1,0.5,1;1,0.5,0)\}\\
R_3 &= \{(0,0,1;1,0,1;1,0.5,1), (1,0,1;1,0.5,1;1,0,0),\\
&\quad (1,0,1;1,0.5,1;1,0.5,0), (0,0,1;1,0,1;1,0.5,1),\\
&\quad (0,0,1;1,0.5,1;1,0,0), (0,0,1;1,0,0;1,0.5,0),\\
&\quad (1,0,1;1,0,0;1,0.5,0), (1,0.5,1;1,0,0;1,0.5,0)\},
\end{aligned}$$

after closing R_2 and R_3 under permutations of "finite" points and permutations of "infinite points." □

[8] "Open" is to be understood with respect to the topology of the affine support of the points under consideration. In Example 3.6, the open convex hull can be obtained by requiring that λ_1, λ_2, and λ_3 are strictly greater than 0. This technique also works in general.

Proposition 5.2 can immediately be generalized:

Proposition 5.4 *Let S be an arbitrary semi-linear set and let $R_1, \ldots, R_{n+1}$ be a finite representation of S. Then*

$$S = \bigcup_{i=1}^{n+1} \bigcup_{((\vec{p}_1,t_1),\ldots,(\vec{p}_i,t_i))\in R_i} \mathrm{CH}((\vec{p}_1,t_1),\ldots,(\vec{p}_i,t_i)),$$

where, for $j = 1,\ldots,i$, $t_j = 1$ or $t_j = 0$ and $\mathrm{CH}(p_1,\ldots,p_i)$ is the the open convex hull of $(\vec{p}_1,t_1),\ldots,(\vec{p}_i,t_i)$, where, for $j = 1,\ldots,i$, $(\vec{p}_j,t_j)$ is interpreted as the "finite" point $\vec{p}_j$, if $t_j = 1$, and as the "infinite" point described by the directional vector $\vec{p}_j$, if $t_j = 0$.

Again, decoding the finite representation of an arbitrary semi-linear set can be done in PFOL in the obvious way, suggested by the formula in the statement of Proposition 5.4.

From now on, we shall understand under *finite representation* any set of relations of the right arity that satisfies Proposition 5.4. (For these sets of relations, the *decoding* query described above always returns the original semi-linear set). The particular finite representation obtained from the *encoding* query described above will be referred to as the *canonical finite representation.*

We now turn back to our motivation for considering finite representations, namely "lifting" query languages defined on finite databases to query languages defined on more general databases.

Let $\mathcal{Q}$ be a query language defined on finite databases. We define *Lift*($\mathcal{Q}$) to be the query language defined on arbitrary linear databases consisting of all compositions of the above PFOL encoding program, a query of $\mathcal{Q}$, and the above PFOL decoding program. The following property is now immediate.

Theorem 5.5 *Let $\mathcal{P}$ be a linear query language defined on arbitrary semi-linear databases that is at least as expressive as PFOL. Let $\mathcal{Q}$ be a query language on finite databases whose expressive power is bounded by $\mathcal{P}$. Then Lift($\mathcal{Q}$) is a query language on arbitrary databases whose expressive power is bounded by $\mathcal{P}$. If, moreover, $\mathcal{Q}$ is complete for the $\mathcal{P}$-expressible queries from finite inputs to finite outputs, then Lift($\mathcal{Q}$) has the same expressive power as $\mathcal{P}$.*

As an application of Theorem 5.5, we let $\mathcal{P}$ be the linear queries expressible in FO + poly, and $\mathcal{Q}$ the syntactically defined fragment of FO + poly exhibited by Benedikt and Libkin [5], of which these authors show it is sound and complete for the FO + poly-expressible queries from finite inputs to finite outputs. By Theorem 5.5, *Lift*($\mathcal{Q}$) is sound and complete for the FO + poly-expressible linear queries. In [9], Dumortier and the present authors also exhibited an—admittedly, artificial—sound and complete query language for the FO + poly-expressible linear queries, mainly to motivate the search for more natural such languages. It can be argued that *Lift*($\mathcal{Q}$) is such a language.

6 The expressive power of PFOL

We are now ready to show that PFOL can express a wide range of natural, linear FO + poly-expressible queries.

Definition 6.1 A FO+poly-expressible linear query is *constructible* if there exists an SPFOL program that computes, from the canonical finite representation of the input, a superset of the active domain of the canonical finite representation of the output. □

We claim that the notion of a constructible linear query is a natural notion, which, to a certain extent, can be seen as a generalization of the notion of domain preservation in the relational model to the context of the linear spatial database model. Indeed, from the equivalence of SPFOL and SafeEuQl (Theorem 4.3), one can argue that constructibility implies that the output can be "assembled" from "material" "constructed" from the "building blocks" of the input; in the relational model, domain preservation means that the output can be "assembled" from the "building blocks," i.e., the entries, of the input.

We give some examples of constructible queries.

Example 6.2 A Boolean query, restricted to linear inputs, is always linear. The canonical representation of its output is its output itself.[9] Hence, the active domain of the canonical finite representation of the output is empty, and, in particular, contained in the active domain of the canonical representation of the input. Thus, all FO + poly-expressible Boolean queries are constructible. In particular, the colinearity query on arbitrary semi-linear sets is constructible. □

Example 6.3 Let S be an arbitrary semi-linear set. Clearly, a bounding box for S is also a bounding box for both the convex closure and the affine support of S. It is now easy to see that a superset of the active domain of the canonical finite representation of the convex closure, respectively, the affine support, of S can be computed from the canonical finite representation of S. As both the convex-closure query and the affine-support query on arbitrary semi-linear sets are FO + poly-expressible, we may conclude that they are also constructible. □

We have the following main result:

Theorem 6.4 *All constructible queries can be computed in PFOL.*

Proof. (Sketch.) From the results in the previous section, it follows that the induced query from the canonical representation of the input to the canonical representation of the output is also FO + poly-expressible. From the canonical representation of the input, we can moreover compute a—finite—superset, say D, of the active domain of the canonical finite representation of the output in SPFOL, whence in PFOL. Now consider one of the FO+poly queries, say $\varphi(x_1,\ldots,x_n)$, required to compute the canonical finite representation of the output from the canonical finite representation of the input. From a result by Benedikt and Libkin [3, 4], it follows that there exists an equivalent formula $\psi(x_1,\ldots,x_n)$ in which all quantified variables range over the active domain (of the input of ψ). From Definition 6.1, it follows that ψ is further equivalent with

$$(\exists p_1 \in D)\ldots(\exists p_n \in D)(x_1 = p_1 \wedge \ldots \wedge x_n = p_n \wedge \psi(p_1,\ldots,p_n)),$$

which obviously defines a PFOL program. Theorem 6.4 now follows immediately from Theorem 5.5. □

The relevance of Theorem 6.4 with respect to the expressive power of PFOL, is that PFOL can compute a wide range of natural, linear queries, while remaining safe for the FO + poly-expressible linear queries. The equivalence of SPFOL, which is closely related to PFOL-fin, with SafeEuQl,

[9] Indeed, the 0-dimensional relations $\{()\}$ and $\{\}$ can be seen as representing the entire 0-dimensional space (which is a single point, which is therefore also a special point) or representing the empty set (the representation of which is the empty set).

a geometrically inspired point-based language, suggests that PFOL, while coordinate-bases just like FO + linear, is geometrically much more meaningful than the former. Finally, it must be observed that Theorem 5.5 implies that, should SPFOL turn out to be equivalent to PFOL-fin, PFOL would *precisely* compute the constructible queries.

7 Directions for future research

We conclude this paper by mentioning two directions for future research. First, as mentioned on several earlier occasions, the precise relationship between SPFOL and PFOL-fin must be established. Second, the expressive power of (S)PFOL can be manipulated in a flexible way. For instance, the square-root construct, which is not needed in the PFOL programs encoding and decoding an arbitrary semi-linear set, could be removed. We believe that the safe language then precisely expresses the constructions with ruler alone on finite sets of points in the plane. Finally, it is also possible to *supplement* the square-root construct with other constructs. By the equivalence of the present version of SPFOL and SafeEuQl, it follows that trisection of an angle is not expressible in SPFOL. It can be shown that this query would become expressible if a cubic-root construct is added to SPFOL. Clearly, the languages obtained by these and similar manipulations deserve further study.

Acknowledgments

During the preparation of this paper, Michael Benedikt and Leonid Libkin spontaneously communicated to us their paper [5] in response to a question we asked them regarding their earlier work. Upon reading their draft, we discovered some parallels between their work and ours, which in turn allowed us to state the result about sound and complete query languages for the FO + poly-expressible queries at the end of Section 5. This result should therefore be considered at least as much their result as it is ours.

Our thanks also extend to an anonymous referee, whose detailed comments allowed us to improve significantly the presentation of this paper.

References

[1] F. Afrati, T. Andronikos, and T. Kavalieros, "On the Expressiveness of First-Order Constraint Languages," in Proc. *2nd Int'l Workshop on Constraint Databases and their Applications* (Delphi, Greece), V. Gaede, A. Brodsky, O. Günther, D. Srivastata, V. Vianu, and M. Wallace, eds., in *LNCS*, vol. 1191, Springer-Verlag, 1996, pp. 105–115.

[2] F. Afrati, S. Cosmadakis, S. Grumbach, and G. Kuper, "Linear Versus Polynomial Constraints in Database Query Languages," in Proc. *2nd Int'l Workshop on Principles and Practice of Constraint Programming* (Rosario, WA), A. Borning, ed., in *LNCS*, vol. 874, Springer-Verlag, 1994, pp. 181–192.

[3] M. Benedikt, G. Dong, L. Libkin, and L. Wong, "Relational expressive power of constraint query languages," in Proc. *15th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems* (Montreal, Canada), 1996, pp. 5–16.

[4] M. Benedikt, and L. Libkin, "Languages for Relational Databases over Interpreted Structures," in Proc. *16th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems* (Tucson, AZ), 1997, pp. 87–98.

[5] M. Benedikt, L. Libkin, "Safe Constraint Queries," *these proceedings*, 1998.

[6] J. Bochnak, M. Coste, and M-F. Roy, *Géométrie algébrique réelle*, Springer-Verlag, Berlin-Heidelberg, 1987.

[7] A. Brodsky and Y. Kornatzky, "The LyriC Language: Querying Constraint Objects," in Proc. *Post-ILPS'94 Workshop on Constraints and Databases* (Ithaca, NY), 1994.

[8] J. Chomicki, D.Q. Goldin, and G.M. Kuper, "Variable Independence and Aggregation Closure," in Proc. *15th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems* (Montreal, Canada), ACM Press, 1997, pp. 68–77.

[9] F. Dumortier, M. Gyssens, D. Van Gucht, L. Vandeurzen, "On the Decidability of Semi-Linearity of Semi-Algebraic Sets and its Implications for Spatial Databases," in Proc. *16th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems* (Tucson, AZ), ACM Press, 1997, pp. 68–77.

[10] S. Grumbach, P. Rigaux, M. Scholl, and L. Segoufin, "DEDALE, A Spatial Constraint Database," in Proc. *6th Int'l Conf. on Database Programming Languages* (Estes Park, CO), 1997, in preparation.

[11] S. Grumbach, J. Su, and C. Tollu, "Linear Constraint Query Languages: Expressive Power and Complexity," in *Logic and Computational Complexity*, D. Leivant, ed., *LNCS*, vol. 960, Springer-Verlag, 1996.

[12] P.C. Kanellakis and D.Q. Goldin, "Constraint Programming and Database Query Languages," in Proc. *2nd Conf. on Theoretical Aspects of Computer Software*, M. Hagiya and J.C. Mitchell, eds., *LNCS*, vol. 789, Springer-Verlag, 1994, pp. 96–120.

[13] P.C. Kanellakis, G.M. Kuper, and P.Z. Revesz, "Constraint Query Languages," *Journal of Computer and System Sciences*, 51:1, 1995, pp. 26–52.

[14] J. Nievergelt and M. Freeston, eds., Special issue on spatial data, *Computer Journal*, 37:1, 1994.

[15] J. Paredaens, B. Kuijpers, G. Kuper, and L. Vandeurzen, "Euclid, Tarski, and Engeler Encompassed," in Proceedings *6th Int'l Conf. on Database Programming Languages* (Estes Park, CO), 1997, in preparation.

[16] J. Paredaens, J. Van den Bussche, and D. Van Gucht, "Towards a Theory of Spatial Database Queries," in Proc. *13th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems* (Minneapolis, MN), 1994. pp. 279–288.

[17] N. Pissinou, R. Snodgrass, R. Elmasri, I. Mumick, T. Özsu, B. Pernici, A. Segef. B. Theodoulidis, and U. Dayal, "Towards an Infrastructure for Temporal Databases," *SIGMOD Record*, 23:1, 1994, pp. 35–51.

[18] L. Vandeurzen, M. Gyssens, and D. Van Gucht, "On the Desirability and Limitations of Linear Spatial Query Languages," in Proc. *4th Symp. on Advances in Spatial Databases* (Portland, OR, August 1995), M. J. Egenhofer and J.R. Herring, eds., in *Lecture Notes in Computer Science*, vol. 951, Springer-Verlag, 1995, pp. 14–28.

[19] L. Vandeurzen, M. Gyssens, and D. Van Gucht, "On Query Languages for Linear Queries Definable with Polynomial Constraints," in Proc. *2nd Int'l Conf. on Principles and Practice of Constraint Programming* (Cambridge, MA, August 1996), E. Freuder, ed., in *LNCS*, vol. 1118, Springer-Verlag, 1996, pp. 468–481.