Complete Geometric Query Languages

Marc Gyssens, Jan Van den Bussche
University of Limburg[*]

Dirk Van Gucht
Indiana University[†]

[*]Dept. WNI, University of Limburg (LUC), Universitaire Campus, B-3590 Diepenbeek, Belgium. E-mail: {gyssens, vdbuss}@luc.ac.be.

[†]Computer Science Department, Indiana University, Bloomington, IN 47405-4101, USA. E-mail: vgucht@cs.indiana.edu

Proposed running head: Complete geometric query languages

Send proofs to:

Marc Gyssens
Limburgs Universitair Centrum
Universitaire Campus
B-3590 Diepenbeek
Belgium
Phone: +32-11-268248
Fax:+32-11-268299
Email: gyssens@luc.ac.be

Abstract

We extend Chandra and Harel's seminal work on computable queries for relational databases to a setting in which also spatial data may be present, using a constraint-based data model. Concretely, we introduce both coordinate-based and point-based query languages that are complete in the sense that they can express precisely all computable queries that are generic with respect to certain classes of transformations of space, corresponding to certain geometric interpretations of spatial data. The languages we introduce are obtained by augmenting basic languages with a while construct. We also show that the respective basic point-based languages are complete relative to the subclass of the corresponding generic queries consisting of those that are expressible in the relational calculus with real polynomial constraints.

1 Introduction

In their seminal work on computable queries for relational databases [3], Chandra and Harel introduced the notion of *computable query* as a computable function from relational databases to relations that is invariant under all permutations of the universe of atomic data elements. The latter criterion, now known as *genericity*, states that queries should preserve database isomorphisms, or, more intuitively, that they should be defined at the logical level of the data in the database. Chandra and Harel then introduced a query language, QL, and proved it complete, in the sense that precisely all computable queries can be expressed in QL.

The purpose of the present paper is to continue Chandra and Harel's work in the setting of spatial databases.

To do so, we work in an adaptation of the relational model, where the universe of atomic data is the set of real numbers, which may represent coordinates of points, and where relations can be infinite. To ensure finite representability, the relations must be elementarily definable in terms of polynomial inequalities. In mathematical terminology, they must be *semi-algebraic*. Our model is thus an instance of the framework of *constraint databases* introduced by Kanellakis, Kuper, and Revesz [7].

As was already pointed out by Paredaens, Van den Bussche, and Van Gucht [11], this framework can be used in two ways. One possibility consists of using the framework in an uninterpreted manner. In order to model spatial data and geometric applications, however, it is necessary to interpret real numbers as coordinates of points in n-dimensional space. In this setting, the universe of atomic data elements are the points of $\mathbf{R}^n$, rather than the real numbers of $\mathbf{R}$. Also in this paper, we shall clearly distinguish between both ways of using constraint databases. For clarity, the uninterpreted constraint database model will be referred to as the *semi-algebraic database model*, whereas the constraint database model in which the atomic data are elements of $\mathbf{R}^n$, interpreted as points in n-dimensional space will be referred to as the *geometric database model*. Clearly, the geometric database model can be embedded in the semi-algebraic database model.

The question of how Chandra and Harel's concept of genericity extends to the geometric database model was already considered by Paredaens, Van den Bussche, and Van Gucht [11]. It makes no sense to require that queries are invariant under all permutations of space, as (i) most of these bear no geometric meaning whatsoever, and (ii) many realistic queries do not preserve arbitrary permutations of space. Instead, a suitably adapted notion of genericity for spatial data should take into account the precise geometric interpretation intended by the application. Now, it is standard mathematical practice to identify a geometry with a group of transformations of space. If the geometric interpretation of the spatial data intended corresponds to a group G of transformations, then a query in the geometric database model will be defined at the intended geometric level if and only if it is invariant under all transformations in G. Such queries are called *G-generic.*

In our search for complete geometric query languages, we start with a study of the underlying semi-algebraic database model. The language most often considered in the semi-algebraic database model is first-order logic augmented with polynomial inequalities, and relation variables of fixed arities, which we denote by FO[$\mathbf{R}$]. We prove that FO[$\mathbf{R}$] augmented with while-loops, a language which we denote by FO[$\mathbf{R}$] + while, yields a complete query language for this model. It is instructive to contrast this result to Chandra and Harel's, who required unranked relation variables, which can hold relations of any arity, to achieve completeness for the language QL.

We then bootstrap this result, which yields complete query languages in the geometric database model under various geometric interpretations. Syntactically, these languages are all identical to FO[$\mathbf{R}$]+while, but, under each geometric interpretation, the semantics of a program is appropriately defined so as to be guaranteed generic.

This is accomplished by working on canonical representations of databases, rather than on the databases themselves.

The approach to finding complete geometric languages just described yields languages with a very artificial semantics. The main underlying reason is of course the mismatch between the nature of the geometric database model, in which the atomic entities are points, and the nature of the languages considered, which have access to the coordinates of points. However, we can obtain much more natural results when we consider first-order query languages that do not have access to the specific coordinates of points but only to the points themselves as atomic entities. Rather than augmenting first-order logic with polynomial inequalities on real numbers, these query languages provide certain built-in geometrical predicates on points, besides relation variables of fixed arities.

We show that, for several geometrically interesting choices of the transformation group G, there exist appropriate point predicates such that first-order logic on points, augmented with the predicates, expresses precisely all G-generic queries expressible in FO[$\mathbf{R}$]. For example, we show that providing the predicate **between**(p, q, r), which is *true* if q lies on the closed line segment between p and r, yields a first-order query language, denoted FO[**between**], that expresses exactly all queries expressible in FO[$\mathbf{R}$] that are generic for affine geometry.

The results describe above are particularly interesting, because G-genericity of FO[$\mathbf{R}$] queries is undecidable for every non-trivial transformation group G [11]. Our proof, which exploits the classical geometrical construction of addition and multiplication, is inspired by the work of Tarski and his collaborators on axiomatizations of elementary geometry [14, 15, 12].

Finally, we consider query languages which augment these point-based languages with relation variables of fixed arities and while-loops. We show that these language are complete geometric query languages under various geometric interpretations. For example, one of our results is that the language FO[**between**] + while is complete for the affine-generic geometric queries.

Complete generic query languages relative to FO[$\mathbf{R}$] were first discovered by Kuijpers, Paredaens, and Suciu [8]. Our results improve upon theirs in the sense that our languages are purely point-based, while the languages of [8] involve both variables ranging over points and variables ranging over real numbers. Papadimitriou, Suciu, and Vianu [10] obtained relative completeness results for point-based query languages in the context of a different type of genericity than the geometric types of genericity we consider here.

For simplicity, we prove our results for purely spatial database models. To be of practical interest, spatial database models need to support both spatial and non-spatial data. We indicate how our results can be extended to this more general setting.

This paper is organized as follows. The semi-algebraic and geometric database models are presented in Section 2. Semi-algebraic and geometric queries and the notion of genericity are reviewed in Section 3. Complete query languages based on $\mathrm{FO}[\mathbf{R}]+\mathrm{while}$ are presented in Section 4. Completeness results for point-based languages relative to various types of geometric queries expressible in $\mathrm{FO}[\mathbf{R}]$ are presented in Section 5. Completeness results for point-based languages relative to various types of arbitrary geometric queries are presented in Section 6. Finally, the extension of our results to the case in which also non-spatial data are present is discussed in Section 7.

2 Semi-algebraic and geometric databases

In this section, we define semi-algebraic and geometric databases.

Both database models are described using the first-order language of the ordered field of the real numbers $(\mathbf{R}, \leq, +, \times, 0, 1)$, i.e., the language $(\leq, +, \times, 0, 1)$. A first-order formula in this language is called a *real formula*. By Tarski's theorem [16], every real formula can effectively be transformed into an equivalent quantifier-free one (equivalent in $\mathbf{R}$). So we can implicitly assume real formulas to be quantifier-free. A consequence of Tarski's theorem is that truth of real sentences in $\mathbf{R}$ is effectively decidable.

Let $k \geq 0$. A subset A of $\mathbf{R}^k$ is *defined* by a real formula $\varphi(x_1, \ldots, x_k)$ if

$$A = \{(a_1, \ldots, a_k) \in \mathbf{R}^k \mid \varphi(a_1, \ldots, a_k)\}.$$

A subset of $\mathbf{R}^k$ is called *semi-algebraic* if it can be defined by a real formula. Rephrased in a vocabulary slightly more expanded than $(\leq, +, \times, 0, 1)$, a semi-algebraic set is a finite union of sets that can be defined by a system of polynomial inequalities with integer coefficients. (In practice, rational coefficients will often be used, too. This does not enlarge the class of sets being considered, as the denominators can be eliminated.)

Example 2.1 Figure 1 shows a heart-shaped semi-algebraic set in $\mathbf{R}^2$, which can be defined as follows:

$$\{(x,y) \mid (x+1)^2 + y^2 \leq 1 \vee (x-1)^2 + y^2 \leq 1 \vee$$
$$(-1 \leq x \leq 1 \wedge y \geq -2 \wedge ((x+1)^2 + (y+2)^2 \geq 1 \vee (x-1)^2 + (y+2)^2 \geq 1))\}.$$

Figure 2 shows another, arrow-shaped, semi-algebraic set in $\mathbf{R}^2$ which can be defined as follows:

$$\{(x,y) \mid (-2 \leq x \leq 1 \wedge x = y) \vee (x + y \geq 2 \wedge 2x - y \leq 2 \wedge 2y - x \leq 2)\}.$$

Since the latter set is defined entirely in terms of *linear* (in)equalities, it is called *semi-linear*. □

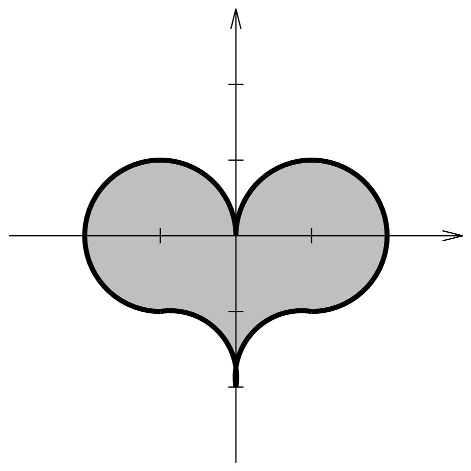

Figure 1: The semi-algebraic set of Example 2.1.

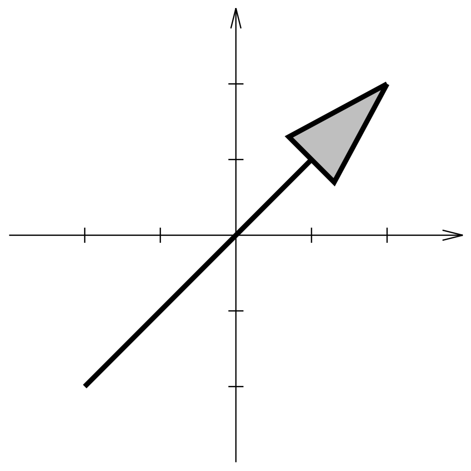

Figure 2: The semi-linear set of Example 2.1.

A *semi-algebraic database* is essentially a store of semi-algebraic sets. To define this formally, we recall that a *relational schema* is a finite set σ of relation names, where each relation name is assigned an arity.

Definition 2.2 Let σ be a relational schema. A *semi-algebraic database* over σ is a structure[1] $\mathcal{D}$ over σ with domain $\mathbf{R}$ such that, for each relation name R of σ, $R^{\mathcal{D}}$ is a semi-algebraic subset of $\mathbf{R}^k$, where k is the arity of R in σ. □

Example 2.3 Let σ be the scheme $\{R, S\}$, wherein both R and S are binary. The structure $\mathcal{D}$ with domain $\mathbf{R}$, with $R^{\mathcal{D}}$ the semi-algebraic set shown in Figure 1 and $S^{\mathcal{D}}$ the semi-linear set shown in Figure 2, is a semi-algebraic database over σ. □

Semantically, a semi-algebraic database can be seen as a relational database, with the exception that the relations may be infinite, as semi-algebraic sets may be infinite. Syntactically, however, a semi-algebraic database can be described finitarily using a (quantifier-free) real formula for each relation name in the schema of the database. We formally define a *representation* of a semi-algebraic database as follows:

Definition 2.4 Let σ be a relational schema and let $\mathcal{D}$ be a semi-algebraic database over σ. A function Φ from the relation names of σ to real formulas is a *representation* of $\mathcal{D}$ if, for each relation name R of σ, $\Phi(R)$ defines $R^{\mathcal{D}}$. □

Example 2.5 Let σ be the scheme $\{R, S\}$, and let $\mathcal{D}$ be the semi-algebraic database considered in Example 2.3. The function which associates with R and S the respective formulas given in Example 2.1 is a representation of $\mathcal{D}$. □

Semi-algebraic databases can be seen as underlying geometric databases, which we define next. From now on, we reserve n to denote the dimension of a geometric space which we shall identify with $\mathbf{R}^n$. Let $k \geq 0$. We shall call a k-ary relation on $\mathbf{R}^n$ semi-algebraic if its image under the canonical bijection[2] between $(\mathbf{R}^n)^k$ and $\mathbf{R}^{nk}$ is a semi-algebraic subset of $\mathbf{R}^{nk}$.

Definition 2.6 Let σ be a relational schema. A *geometric database* over σ in $\mathbf{R}^n$ is a structure $\mathcal{D}$ over σ with domain $\mathbf{R}^n$ such that, for each relation name R of σ, $R^{\mathcal{D}}$ is semi-algebraic. □

A geometric database $\mathcal{D}$ over σ in $\mathbf{R}^n$ can be viewed naturally as a semi-algebraic database $\overline{\mathcal{D}}$ over the schema $\overline{\sigma}$, which has, for each relation name R of σ, a relation name $\overline{R}$ with arity kn, where k is the arity of R in σ. For each relation name R, of arity k, $\overline{R}^{\overline{\mathcal{D}}}$ is obtained from $R^{\mathcal{D}}$ by applying the canonical bijection between $(\mathbf{R}^n)^k$ and $\mathbf{R}^{nk}$.

[1] *Structure* is used here in the sense of mathematical logic [4]. A structure associates, with each of its relation names, a relation of the appropriate arity over the domain of the structure.

[2] The *canonical bijection* between $(\mathbf{R}^n)^k$ and $\mathbf{R}^{nk}$ associates with each k-tuple $(\mathbf{x}_1, \ldots, \mathbf{x}_k)$ of $\mathbf{R}^n$ the nk-tuple $(\mathbf{x}_1^1, \ldots \mathbf{x}_n^1, \ldots, \mathbf{x}_1^k, \ldots, \mathbf{x}_n^k)$, where for $1 \leq i \leq k$ and $1 \leq j \leq n$, $\mathbf{x}_j^i$ denotes the j-th component of $\mathbf{x}_i$.

Example 2.7 The database defined in Example 2.3 can be seen as the underlying semi-algebraic database for a geometric database in $\mathbf{R}^2$, consisting of two unary relations (i.e., sets) of points in the plane. $\square$

3 Semi-algebraic and geometric queries

In this section, we define algebraic and geometric queries and review a classification for geometric queries based on genericity types.

Definition 3.1 Let σ be a relational schema. A k-ary *semi-algebraic query* Q over σ is a partial function on the set of semi-algebraic databases over σ, satisfying the following conditions:

1. for each semi-algebraic database $\mathcal{D}$ over σ on which Q is defined, $Q(\mathcal{D})$ is a semi-algebraic subset of $\mathbf{R}^k$; and
2. there is an algorithm taking representations of semi-algebraic databases as input, and returning real formulas as output, satisfying the following conditions:
   (a) for each semi-algebraic database $\mathcal{D}$ over σ, and for each representation Φ of $\mathcal{D}$, the algorithm terminates on input Φ if and only if Q is defined on $\mathcal{D}$; and
   (b) for each semi-algebraic database $\mathcal{D}$ on which Q is defined, and for each representation Φ of $\mathcal{D}$, the output of the algorithm on Φ is a real formula defining $Q(\mathcal{D})$. $\square$

The second condition in Definition 3.1 indicates in which sense semi-algebraic queries are computable.

Example 3.2 Let $\sigma = \{R, S\}$ be the schema defined in Example 2.3. The following are examples of semi-algebraic queries over σ:

1. "Compute the projection onto the x-axis of the semi-algebraic set associated with R" is a unary semi-algebraic query.
2. "Find the intersection of the semi-algebraic sets associated with R and S" is a binary semi-algebraic query.
3. "Decide whether the semi-algebraic set associated with R is topologically connected" is a null-ary[3] semi-algebraic query.
4. "Compute the convex hull[4] of the semi-linear set associated with S" is a binary semi-algebraic query.

[3]The null-ary semi-algebraic sets, $\emptyset$ and $\{()\}$, can be interpreted as the Boolean values *false* and *true*, respectively.

[4]The *convex hull* of a set S is the smallest convex set containing S.

5. "Decide whether the semi-algebraic set associated with R is a circle" is a null-ary semi-algebraic query.

6. "Decide whether there is a point in the semi-algebraic set associated with R and a point in the semi-linear set associated with S at distance 1 from each other" is a null-ary semi-algebraic query.

7. For the query to follow, we consider, besides R and S, a third relation scheme T of arity 2. "Decide whether each of the relations R, S, and T a is singleton such that the triple of points (u, v, w), with $u \in R$, $v \in S$, and $w \in T$, is a positively oriented orthonormal basis[5] of $\mathbf{R}^2$" is a null-ary semi-algebraic query.

8. "Compute the left-most points of the semi-algebraic set associated with R" is a binary semi-algebraic query.

9. "Decide whether the semi-linear set associated with S contain the point $(0, 0)$" is a null-ary semi-algebraic query. $\square$

In analogy to Definition 3.1, we define *geometric queries*.

Definition 3.3 Let σ be a relational schema. A k-ary *geometric query* Q over σ in $\mathbf{R}^n$ is a partial function on the set of geometric databases over σ, satisfying the following conditions:

1. for each geometric database $\mathcal{D}$ over σ on which Q is defined, $Q(\mathcal{D})$ is a semi-algebraic subset of $(\mathbf{R}^n)^k$; and

2. Q is computable in the sense of Definition 3.1 (where "semi-algebraic" is replaced by "geometric"). $\square$

Example 3.4 Consider again the queries in Example 3.2. Assume that we work in the plane, i.e., in $\mathbf{R}^2$.

- Query 1 is *not* an example of a geometric query.

- Queries 3, 5, 6, 7, and 9 are examples of null-ary geometric queries.

- Queries 2, 4, and 8 are examples of unary geometric queries. $\square$

[5] A *basis* of $\mathbf{R}^n$ is an $(n+1)$-tuple of points $(o, e_1, \ldots, e_n)$ such that the vectors $\overrightarrow{oe_1}$ through $\overrightarrow{oe_n}$ are linearly independent. A basis is *orthogonal* if the vectors $\overrightarrow{oe_1}$ through $\overrightarrow{oe_n}$ are pairwise orthogonal. A basis is *orthonormal* if it is orthogonal and the vectors $\overrightarrow{oe_1}$ through $\overrightarrow{oe_n}$ have unit length. A basis is *positively oriented* if it has the same orientation as the standard basis of $\mathbf{R}^n$, which is the case if the determinant of the $n \times n$ matrix consisting of the components of the vectors $\overrightarrow{oe_1}$ through $\overrightarrow{oe_n}$ is positive; it is *negatively oriented* otherwise.

Since geometric databases can be identified with certain kinds of semi-algebraic databases, a comparison of Definitions 3.1 and 3.3 reveals that geometric queries can be identified with certain kinds of semi-algebraic queries.

In the geometric database model, the result of many natural queries does not depend on the particular coordinates assigned to points by the canonical coordinate system in the space considered. More precisely, natural geometric queries preserve coordinate system transitions. The coordinate transitions that must be considered, of course, depend on the geometry of the space, which can be described by a group of transformations (permutations) of space. Therefore, we adopt the following general notion of genericity, originally proposed by Paredaens, Van den Bussche, and Van Gucht [11].

Definition 3.5 Let σ be a relational schema and let Q be a geometric query over σ in $\mathbf{R}^n$, and let G be a group of transformations of $\mathbf{R}^n$. Then Q is called *G-generic* if, for any two geometric databases $\mathcal{D}$ and $\mathcal{D}'$ over σ in $\mathbf{R}^n$ for which $\mathcal{D}' = g(\mathcal{D})$ for some transformation g in G, we have that $Q(\mathcal{D}') = g(Q(\mathcal{D}))$. □

In affine geometry, for instance, G is the group of affinities, i.e., compositions of linear transformations and translations, and the corresponding class of queries is called the affine-generic queries. In two-dimensional affine geometry, it would make no sense to ask for all points in the database lying in the unit disk, as this is not an affine-generic query. (Points can be moved in and out of the unit disk by applying a translation, which is an affine transformation). It would make sense, however, to ask for all straight lines contained in the database, as this query is affine-generic: collinearity is preserved under affinities.

Besides *affine genericity*, there are several other notions of genericity that correspond to sensible geometry. We summarize some of them below:

- *Similarity genericity*, with respect to the group of similarities, i.e., compositions of isometries (see below) and scalings. This genericity notion corresponds to Euclidean geometry.

- *Isometry genericity*, with respect to the isometries, i.e., compositions of translations, rotations, and reflections. This genericity notion corresponds to the fragment of Euclidean geometry where absolute rather than relative measures are important.

- *Direct-isometry genericity*, with respect to the direct isometries, i.e., compositions of translations and rotations. This genericity notion corresponds to the fragment of the previous geometry where also orientation is important. In this geometry, two objects are considered isomorphic if one can be mapped to the other by a rigid motion.[6]

[6]A rigid motion is a transformation that can be specified as a composition of translations and rotations [5].

- *Translation genericity*, with respect to the translations. This genericity notion corresponds to the fragment of the previous geometry where the relative position of objects (e.g., in the two-dimensional case, above or left of) is important.

Example 3.6 Consider again the queries in Example 3.2.

- Queries 2, 3, and 4 are affine-generic geometric queries.

  Query 2 is affine-generic, because affine transformations are permutations, and the concept of intersection of a pair of sets is preserved under permutations.

  Query 3 is affine-generic, because affine transformations are homeomorphisms, which preserve topological connectedness [9].

  To see that Query 4 is affine-generic, let U and V be an arbitrary pair of sets in the plane for which there exists an affinity g such that $g(U) = V$. We need to show that $g(\text{convexhull}(U)) = \text{convexhull}(V)$. We show that $g(\text{convexhull}(U)) \subseteq \text{convexhull}(V)$. (The reverse inclusion holds because g^{-1} is also an affinity.) Let p be an arbitrary point in convexhull(U). Then $p = \lambda_1 p_1 + \lambda_2 p_2 + \lambda_3 p_3$, with p_1, p_2, and p_3 points in U, $\lambda_1 \geq 0$, $\lambda_2 \geq 0$, and $\lambda_3 \geq 0$, and $\lambda_1 + \lambda_2 + \lambda_2 = 1$. Since g is an affinity, there exist real numbers a, b, c, d, e, and f with $ad - bc \neq 0$ such that, for each point $q = (x, y)$, $g(q) = (ax + by + e, cx + dy + f)$. From this information, it is a simple algebraic exercise to determine that $g(p) \in \text{convexhull}(V)$.

- Query 5 is a similarity-generic query that is not affine-generic.

  Query 5 is similarity-generic, because similarities are defined to be exactly those transformations that preserve shape. Hence, if U and V are sets such that $g(U) = V$, with g a similarity, then either U and V are both circles, or neither of them are circles.

  Query 5 is not affine-generic, however. To see this, let $U = \{(x, y) \mid x^2 + y^2 = 1\}$ and $V = \{(x, y) \mid 4x^2 + y^2 = 1\}$. Clearly, U is a circle and V is an ellipse that is not a circle, yet the affinity $g(x, y) = (x, 2y)$ maps U to V.

- Query 6 is an isometry-generic query that is not similarity-generic.

  Query 6 is isometry-generic, because isometries are defined to be exactly those transformations that preserve distance. Hence, if $U = (U_1, U_2)$ and $V = (V_1, V_2)$ are geometric databases such that $g(U) = V$, with g an isometry, then either U and V both satisfy the distance condition in the query, or neither of them does.

  Query 6 is not similarity-generic, however. To see this, let U and V be the databases $(\{(0, 0)\}, \{(0, 1)\})$ and $(\{(0, 0)\}, \{(0, 2)\})$ respectively. Clearly, U satisfies the condition of the query and V does not, yet the similarity $g(x, y) = (2x, 2y)$ maps U to V.

- Query 7 is a direct-isometry-generic query that is not isometry-generic.

  To see that Query 7 is a direct-isometry-generic query, let $U = (U_1, U_2, U_3)$ and $V = (V_1, V_2, V_3)$ be two geometric databases, and let g be a direct isometry such that $g(U) = V$. If U satisfies the condition of the query, then U can be interpreted as a positively oriented orthonormal basis. Since direct isometries preserve distance, orthogonality, and orientation, V also satisfies the condition of the query.

To see that Query 7 is not isometry-generic, let $U = (\{(0,0)\}, \{(1,0)\}, \{(0,1)\})$ and $V = (\{(0,0)\}, \{(1,0)\}, \{(0,-1)\})$. Clearly, U is mapped to V by the reflection[7] with respect to the x-axis. Since U represents the standard basis, it satisfies the condition of the query. However, V represents a negatively oriented basis, and, therefore, does not satisfy the condition of the query.

- Query 8 is a translation-generic query that is not direct-isometry-generic.

  Query 8 is translation-generic, because translations preserve the concept of "being to the left of."

  The query is not direct-isometry-generic, however. To see this, let $U = \{(0,0),(1,0)\}$ and $V = \{(0,0),(-1,0)\}$. The reflection with respect to the origin[8] maps U to V. However, the leftmost point of U, which is $(0,0)$, is not mapped by this rotation to the leftmost point of V, which is $(-1,0)$.

- Query 9 is a geometric query that is not translation-generic.

  To see that Query 9 is not translation-generic, let $U = \{(0,0)\}$ and $V = \{(1,0)\}$. Clearly, the translation $g(x,y) = (x+1,y)$ maps U to V, However, the origin $(0,0)$ is in U, but not in V. $\square$

4 Complete languages for semi-algebraic queries

In this section, we consider the query languages FO[**R**] and FO[**R**] + while and show that the latter language expresses exactly all semi-algebraic queries. For a wide variety of geometries, we then show how the semantics of programs in this language can be modified so as to be guaranteed generic, yielding query languages expressing exactly all generic geometric queries of the type considered.

4.1 Semi-algebraic queries

Let σ be a relational schema. A first-order formula $\varphi(x_1,\ldots,x_k)$ in the language of the real numbers augmented with the relation names of σ defines on each semi-algebraic database $\mathcal{D}$ over σ a subset $\varphi(\mathcal{D})$ of $\mathbf{R}^k$ in the standard manner. Since $\varphi(\mathcal{D})$ is obviously semi-algebraic, φ thus defines a k-ary semi-algebraic query over σ. The basic query language obtained by all such formulas φ is denoted by FO[**R**].

[7]Reflections are isometries.

[8]Reflections with respect to a point are rotations around that point over an angle of 180°, and, therefore, are rigid motions.

Example 4.1 Consider again the queries introduced in Example 3.2.

Each of these queries, except for the connectivity query (Query 3), is expressible in FO[**R**].[9]

- Query 1 is expressed as $\{(x) \mid (\exists y) R(x,y)\}$.
- Query 2 is expressed as $\{(x,y) \mid R(x,y) \wedge S(x,y)\}$.
- Query 4 is expressed as

  $$\{(x,y) \mid (\exists x_1)(\exists y_1)(\exists x_2)(\exists y_2)(\exists x_3)(\exists y_3)(\exists \lambda_1)(\exists \lambda_2)(\exists \lambda_3)(S(x_1,y_1) \wedge S(x_2,y_2) \wedge S(x_3,y_3) \wedge \\ \lambda_1 \geq 0 \wedge \lambda_2 \geq 0 \wedge \lambda_3 \geq 0 \wedge \lambda_1 + \lambda_2 + \lambda_3 = 1 \wedge \\ x = \lambda_1 x_1 + \lambda_2 x_2 + \lambda_3 x_3 \wedge y = \lambda_1 y_1 + \lambda_2 y_2 + \lambda_3 y_3)\},$$

  i.e., a point (x,y) is in the convex hull of S if it can be written as a convex combination of three points (x_1,y_1), (x_2,y_2), and (x_3,y_3) in S.
- Query 5 is expressed as

  $$(\exists a)(\exists b)(\exists r)(r > 0 \wedge (\forall x)(\forall y)(R(x,y) \Leftrightarrow (x-a)^2 + (y-b)^2 = r^2)).$$
- Query 6 is expressed as

  $$(\exists x_1)(\exists y_1)(\exists x_2)(\exists y_2)(R(x_1,y_1) \wedge S(x_2,y_2) \wedge (x_2 - x_1)^2 + (y_2 - y_1)^2 = 1).$$
- Query 7 is expressed as

  $$(\exists a_{11})(\exists a_{12})(\exists a_{21})(\exists a_{22})(\exists b_1)(\exists b_2)(a_{11}a_{22} - a_{12}a_{21} = 1 \\ \wedge (\forall x)(\forall y)(R(x,y) \Leftrightarrow S(a_{11}x + a_{12}y + b_1, a_{21}x + a_{22}y + b_2))).$$

  Note that the variables a_{11}, a_{12}, a_{21}, a_{22}, b_1, b_2, are used to represent the rigid motion that maps the point (x,y) to the point $(a_{11}x + a_{12}y + b_1, a_{21}x + a_{22}y + b_2)$. The formula then expresses that the sets R and S can be mapped to each other by a direct isometry, i.e., a rigid motion.
- Query 8 is expressed as $\{(x,y) \mid R(x,y) \wedge \neg(\exists x')(\exists y')(R(x',y') \wedge x' < x)\}$.
- Query 9 is expressed as $S(0,0)$. □

We can extend FO[**R**] into a full-fledged programming language, which we denote by FO[**R**] + while.

A *program* over σ is a finite sequence of *statements* and *while-loops*. Each statement has the form $R := \{(x_1, \ldots, x_k) \mid \varphi(x_1, \ldots, x_k)\}$, with R a relation variable of arity k and φ a first-order formula in the language of the real numbers augmented with the relation names of σ and the previously introduced relation variables. Each while-loop

[9] By the combined results of Grumbach and Su [6] and of Benedikt, Dong, Libkin, and Wong [2], the connectivity query is not expressible in FO[**R**]. From results by Schwartz and Sharir [13], it follows, however, that this query is computable.

has the form **while** φ **do** P, where P is a program and φ is a first-order sentence in the language of the real numbers augmented with the relation names of σ and the previously introduced relation variables.

Semantically, a program in the query language FO[**R**] + while expresses a semi-algebraic query in the obvious way as soon as one of its relation variables has been designated as the output variable. Of course, since while-loops need not terminate, this query will in general not be totally defined (as is the case with FO[**R**] queries).

As announced, we can show that FO[**R**]+while is complete for the semi-algebraic queries.

Theorem 4.2 *Every semi-algebraic query is expressible in* FO[**R**] + while.

Proof. Let Q be a k-ary semi-algebraic query over a schema σ. Let K be the maximum of k and the arities of relation names of σ. Then every relation in a semi-algebraic database over σ can be defined by a quantifier-free real formula using only the variables $x_1, \ldots, x_K$.

We next introduce a specific way of encoding such formulas as natural numbers in such a way that that the encoding of a subterm or subformula occurring in another term or formula comes before the encoding of that term or formula. Notice that these formulas, and the terms that can occur in them, are strings over the finite alphabet $\Sigma = \{a_1, \ldots, a_{9+K}\}$, where the alphabet symbols are shown in Table 1. Any string $(a_{i_1} \ldots a_{i_n})$ over Σ can be encoded as a natural number $\text{enc}(a_{i_1} \ldots a_{i_n}) = p_1^{i_1} \ldots p_n^{i_n}$, where p_j is the j-th prime number. Observe that if s is a substring of t, then $\text{enc}(s) < \text{enc}(t)$.

a_1	(
a_2	)
a_3	$\neg$
a_4	$\vee$
a_5	$\leq$
a_6	$+$
a_7	$\times$
a_8	0
a_9	1
a_{10}	x_1
$\vdots$	
a_{9+K}	x_K

Table 1: The alphabet Σ of formulas and terms.

Now let R be a relation name of arity l in σ. We show that there is a program Encode_R which, when applied to a database $\mathcal{D}$, computes in the variable n_R the encoding of a real formula defining $R^{\mathcal{D}}$.

To do so, we point out that programs in FO[**R**] + while have full computational power on natural numbers. Indeed, natural numbers can be stored in variables in the form of unary singleton relations, and it is easy to simulate counter programs. The program

```
$n := 0$; $T := \emptyset$; $F := \emptyset$;
$Found := false$;
**while** $\neg Found$ **do**
  $n := n + 1$;
  **if** $n$ encodes $x_1$ **then**
    $T := T \cup \{(n, a_1, \ldots, a_l, a_1) \mid a_1, \ldots, a_l \in \mathbf{R}\}$ **else**
      $\vdots$
  **if** $n$ encodes $x_l$ **then**
    $T := T \cup \{(n, a_1, \ldots, a_l, a_l) \mid a_1, \ldots, a_l \in \mathbf{R}\}$ **else**
  **if** $n$ encodes 0 **then**
    $T := T \cup \{(n, a_1, \ldots, a_l, 0) \mid a_1, \ldots, a_l \in \mathbf{R}\}$ **else**
  **if** $n$ encodes 1 **then**
    $T := T \cup \{(n, a_1, \ldots, a_l, 1) \mid a_1, \ldots, a_l \in \mathbf{R}\}$ **else**
  **if** $n$ encodes $(s + t)$ **then**
    $T := T \cup \{(n, a_1, \ldots, a_l, c + d) \mid T(\mathrm{enc}(s), a_1, \ldots, a_l, c) \wedge T(\mathrm{enc}(t), a_1, \ldots, a_l, d)\}$ **else**
  **if** $n$ encodes $(s \times t)$ **then**
    $T := T \cup \{(n, a_1, \ldots, a_l, cd) \mid T(\mathrm{enc}(s), a_1, \ldots, a_l, c) \wedge T(\mathrm{enc}(t), a_1, \ldots, a_l, d)\}$ **else**
  **if** $n$ encodes $(s \leq t)$ **then**
    $F := F \cup \{(n, a_1, \ldots, a_l) \mid (\exists c)(\exists d)(T(\mathrm{enc}(s), a_1, \ldots, a_l, c) \wedge$
    $T(\mathrm{enc}(t), a_1, \ldots, a_l, d) \wedge c \leq d)\}$ **else**
  **if** $n$ encodes $(\neg\varphi)$ **then**
    $F := F \cup \{(n, a_1, \ldots, a_l) \mid \neg F(\mathrm{enc}(\varphi), a_1, \ldots, a_l)\}$ **else**
  **if** $n$ encodes $(\varphi \vee \psi)$ **then**
    $F := F \cup \{(n, a_1, \ldots, a_l) \mid F(\mathrm{enc}(\varphi), a_1, \ldots, a_l) \vee F(\mathrm{enc}(\psi), a_1, \ldots, a_l)\}$;
  $Found := n$ encodes a formula $\wedge (\forall a_1) \ldots (\forall a_l)(F(n, a_1, \ldots, a_l) \Leftrightarrow R(a_1, \ldots, a_l))$
**od**;
$n_R := n$.
```

Figure 3: The program *Encode*$_R$.

Encode$_R$, shown in Figure 3, builds up relations T (for term) and F (for formula). The arity of T is $l+2$; each tuple in T is of the form $(t, a_1, \ldots, a_l, \tau)$, where t is the encoding of a term which *only* uses the variables $x_1, \ldots, x_l$, where $a_1, \ldots, a_l$ are real numbers, and where τ is the value of t evaluated under the valuation $x_1 \mapsto a_1, \ldots, x_l \mapsto a_l$. The arity of F is $l + 1$; each tuple in F is of the form $(f, a_1, \ldots, a_l)$, where f is the encoding of a formula which *only* uses the variables $x_1, \ldots, x_l$, and where $f(a_1, \ldots, a_l)$ is *true*.

The program *Encode*$_R$ works, because encodings and decodings can be performed effectively, and because terms and formulas are evaluated before the terms and formulas in which they occur.

We define the program *Encode* as the composition of all programs *Encode*$_R$ for all relation names R of σ.

We next show that there exists a program *Decode* which, when applied to the encoding f of a formula φ, computes in a relation variable *Dec* the semi-algebraic subset of $\mathbf{R}^k$ defined by φ. Thereto, it suffices to modify the program *Encode*$_R$ in Figure 3 as follows. First, substitute the subscript k for the subscript l in Encode_R. Next, the statement assigning the variable *Found* in the body of the while-loop is replaced by $\mathit{Found} := n = f$. The last statement of the program is replaced by $\mathit{Dec} := \{(a_1, \ldots, a_k) \mid F(n, a_1, \ldots, a_k)\}$.

Now, revisiting the k-ary computable query Q over σ, denote the set of relation names of σ by $\{R_1, \ldots, R_r\}$. Then there exists a counter program M such that, for each database $\mathcal{D}$ on which Q is defined, if $(n_{R_1}, \ldots, n_{R_r})$ are the results of applying program *Encode* to $\mathcal{D}$, then $M(n_{R_1}, \ldots, n_{R_r})$ is the encoding of a quantifier-free formula defining $Q(\mathcal{D})$, using the variables $x_1, \ldots, x_k$. If $Q(\mathcal{D})$ is not defined, then M does not halt on this input. As already noted above, we can simulate M by a program P in $\mathrm{FO}[\mathbf{R}] + \mathrm{while}$. Hence, query Q is expressed by the program

$$\mathit{Encode};\ P;\ \mathit{Decode}. \qquad \square$$

Recently, a lot of attention has been devoted to *semi-linear databases*, which are essentially stores of semi-linear sets, and languages to query these databases. Semi-algebraic queries on semi-linear databases returning semi-linear outputs are called *semi-linear queries*. To design languages for expressing semi-linear queries, it is natural to consider, as a core language, the language $\mathrm{FO}[\mathbf{R}_{\mathrm{lin}}]$, which is $\mathrm{FO}[\mathbf{R}]$ restricted to formulas in which only linear polynomials occur. From re-examining the proof of Theorem 4.2, the following is readily derived.

Corollary 4.3 *Every semi-linear query is expressible in* $\mathrm{FO}[\mathbf{R}_{\mathrm{lin}}] + \mathrm{while}$.

In fact, the proof is the same as that of Theorem 4.2, with the exception that every statement in that proof which refers to multiplication must be omitted.

4.2 Geometric queries

We will assume that we work in the n-dimensional space $\mathbf{R}^n$, $n \geq 1$.

Let σ be a relational schema, and let G be a group of transformations of $\mathbf{R}^n$. Representations of geometric databases over σ are essentially strings over some finite alphabet and hence can be compared lexicographically. We can thus define the following notions.

Definition 4.4
1. Two geometric databases $\mathcal{D}$ and $\mathcal{D}'$ are called *G-isomorphic* if $\mathcal{D}' = g(\mathcal{D})$ for some g in G.

2. The *G-canonization* of a geometric database $\mathcal{D}$, denoted by $\mathit{canon}_G(\mathcal{D})$, is the geometric database $\mathcal{D}'$ which is G-isomorphic to $\mathcal{D}$ and has a representation that occurs lexicographically first among the representations of geometric databases G-isomorphic to $\mathcal{D}$.

3. The *G-type* of $\mathcal{D}$, denoted by $\mathit{Type}_G(\mathcal{D})$, equals $\{g \in G \mid g(\mathcal{D}) = \mathit{canon}_G(\mathcal{D})\}$. $\square$

Clearly, if $\mathcal{D}$ and $\mathcal{D}'$ are G-isomorphic databases, then $canon_G(\mathcal{D}) = canon_G(\mathcal{D}')$.

Canonization can be carried out effectively for a wide variety of transformation groups G. It suffices for G to be identifiable with a semi-algebraic subset of $\mathbf{R}^\ell$, for some fixed ℓ, such that the "graph" of G, the set $\{(g, x_1, \ldots, x_n, x'_1, \ldots, x'_n) \mid g \in G \wedge g(x_1, \ldots, x_n) = (x'_1, \ldots, x'_n)\}$ is a semi-algebraic subset of $\mathbf{R}^{\ell+2n}$. If this is the case, we call G *semi-algebraic.* For semi-algebraic transformation groups G, we can compute a representation of $canon_G(\mathcal{D})$ from a representation of $\mathcal{D}$ by enumerating all representations of databases $\mathcal{D}'$ until we find one for which $(\exists g \in G)(g(\mathcal{D}) = \mathcal{D}')$ is *true.* This condition is a real sentence, and, therefore, decidable by Tarski's theorem.

Example 4.5 Most naturally occurring transformation groups are semi-algebraic; in particular, all transformation groups considered in Section 3 are.

The group of affinities is semi-algebraic. An affinity is a composition of a linear transformation, which can be described by a regular $n \times n$ matrix, and a translation, which can be described by an n-dimensional vector. If $n = 2$, we can identify the group of affinities with the semi-algebraic set

$$\{(a_{11}, a_{12}, a_{21}, a_{22}, b_1, b_2) \mid a_{11}a_{22} - a_{12}a_{21} \neq 0\},$$

whence the graph of this group equals

$$\begin{aligned}&\{(a_{11}, a_{12}, a_{21}, a_{22}, b_1, b_2, x, y, x', y') \mid a_{11}a_{22} - a_{12}a_{21} \neq 0 \wedge \\ &x' = a_{11}x + a_{12}y + b_1 \wedge y' = a_{21}x + a_{22}y + b_2\},\end{aligned}$$

which is clearly a semi-algebraic set.

The group of similarities is semi-algebraic. If $n = 2$, the graph of this group equals

$$\begin{aligned}&\{(k, a, b, c, d, x, y, x', y') \mid k^2 = 1 \wedge a^2 + b^2 \neq 0 \wedge \\ &x' = ax - by + c \wedge y' = kbx + kay + d\},\end{aligned}$$

which is clearly a semi-algebraic set.

The group of isometries is semi-algebraic. For $n = 2$, we only need to refine the case for similarities by insisting that $a^2 + b^2 = 1$.

The group of direct isometries is semi-algebraic. For $n = 2$, we only need to refine the case for similarities by insisting that $a^2 + b^2 = 1$ and $k = 1$.

The group of translations is semi-algebraic. For $n = 2$, we only need to refine the case for similarities by insisting that $a = 1$, $b = 0$, and $k = 1$. □

We are now ready to define a modified semantics of programs, in conjunction with some semi-algebraic group G. If P is a program expressing a geometric query and $\mathcal{D}$ a database, then we define

$$P^G(\mathcal{D}) := \bigcup \{g^{-1}(P(canon_G(\mathcal{D}))) \mid g \in Type_G(\mathcal{D})\}.$$

We can show the following.

Theorem 4.6 *The partial function mapping $\mathcal{D}$ to $P^G(\mathcal{D})$ for each geometric database $\mathcal{D}$ is a G-generic geometric query (in particular, it is computable). Moreover, if P already expresses a G-generic query then $P^G(\mathcal{D}) = P(\mathcal{D})$ for each geometric database $\mathcal{D}$.*

Proof. We first prove that the mapping P^G is a G-generic geometric query. The computability of P^G follows from the arguments made above. We can thus concentrate on the G-genericity. Let $\mathcal{D}$ and $\mathcal{D}'$ be geometric databases such that $\mathcal{D}' = g(\mathcal{D})$ for some transformation g in G. We have to prove that $P^G(\mathcal{D}') = g(P^G(\mathcal{D}))$.

Since $\mathcal{D}' = g(\mathcal{D})$, and also $canon_G(\mathcal{D}') = canon_G(\mathcal{D})$, we have furthermore that

$$Type_G(\mathcal{D}') = Type_G(\mathcal{D}) \circ g^{-1},$$

where composition must be interpreted element-wise. Thus we can deduce that

$$\begin{aligned} g(P^G(\mathcal{D})) &= g\Big(\bigcup\{h^{-1}(P(canon_G(\mathcal{D}))) \mid h \in Type_G(\mathcal{D})\}\Big) \\ &= \bigcup\{g \circ h^{-1}(P(canon_G(\mathcal{D}))) \mid h \in Type_G(\mathcal{D})\} \\ &= \bigcup\{(h \circ g^{-1})^{-1}(P(canon_G(\mathcal{D}))) \mid h \in Type_G(\mathcal{D})\} \\ &= \bigcup\{h'^{-1}(P(canon_G(\mathcal{D}))) \mid h' \in Type_G(\mathcal{D}')\} \\ &= P^G(\mathcal{D}'). \end{aligned}$$

If P itself is G-generic, then, for each g in $Type_G(\mathcal{D})$,

$$g^{-1}(P(canon_G(\mathcal{D}))) = g^{-1}(P(g(\mathcal{D}))) = g^{-1} \circ g(P(\mathcal{D})) = P(\mathcal{D}),$$

whence $P^G = P$. □

According to Theorem 4.6, we can produce complete, generic, geometric query languages for a wide variety of geometries. Notice that all these languages are syntactically identical to FO[$\mathbf{R}$] + while, and are thus very artificial. In Section 6, we provide more natural languages which are sound and complete for the various classes of computable geometric queries. Some of the groundwork to establish these results will be laid in the following section.

5 First-order geometric query languages

In this section, we first propose a family of first-order query languages, FO[Π], parameterized by sets Π of so-called point predicates. We then proceed by identifying several members of this family and showing that each of these is sound and complete for a natural genericity class of geometric queries expressible in FO[$\mathbf{R}$].

We recall that the domain of a geometric database in dimension n is $\mathbf{R}^n$, i.e., the geometric space itself, the elements of which we naturally call *points*. In the logic-based query languages we will define next, the variables stand for points (as opposed to real numbers). Thus the predicates used in these languages are evaluated over the set of points of $\mathbf{R}^n$ (as opposed to the set of real numbers) and will therefore be referred to as *point predicates.*

Apart from relation names, we consider the following point predicates[10]:

- **between**(p_1, p_2, p_3), which is *true* if either p_2 lies on the closed line segment between p_1 and p_3, or if p_1, p_2, and p_3 coincide;
- **equidistance**(p_1, p_2, p_3, p_4), which is *true* if the distance between p_1 and p_2 equals the distance between p_3 and p_4;
- **unitdistance**(p_1, p_2), which is *true* if the distance between p_1 and p_2 equals 1;
- **positive**$(o, p_1, \ldots, p_n)$, which is *true* if $(o, p_1, \ldots, p_n)$, is a positively oriented basis of $\mathbf{R}^n$;

  oriented,
- **smaller**$_i(p_1, p_2)$, $1 \leq i \leq n$, which is *true* if the i-th component of p_1 is smaller than the i-th component of p_2.

Now, let Π be a finite set of point predicates such as the ones above, and let σ be a relational schema. A first-order formula

$\varphi(\widehat{v}_1, \ldots, \widehat{v}_k)$ over the relation names of σ and the predicate names in Π defines on each geometric database $\mathcal{D}$ over σ a subset $\varphi(\mathcal{D})$ of $(\mathbf{R}^n)^k$ in the standard manner.[11] Notice that variables now range over $\mathbf{R}^n$ instead of $\mathbf{R}$, i.e., over points instead of coordinates. If the predicates in Π can be defined by real formulas (in terms of the coordinates of the points involved, φ is equivalent to an FO[$\mathbf{R}$]-formula $\overline{\varphi}$ over the schema $\overline{\sigma}$ corresponding to σ (cf. Section 2). Hence, $\varphi(\mathcal{D})$ will be semi-algebraic and thus φ defines a k-ary geometric query over σ. The query language obtained is denoted by FO[Π].

We observe that all point predicates considered above are definable by real formulas. To illustrate this, we assume that we work in the plane. Furthermore, the first and second coordinates of a point p will be denoted by p^1 and p^2, respectively.

1. The real formula

   $$(\exists\lambda_1)(\exists\lambda_2)(p_2^1 = \lambda_1 p_1^1 + \lambda_2 p_3^1 \wedge p_2^2 = \lambda_1 p_1^2 + \lambda_2 p_3^2 \wedge \lambda_1 \geq 0 \wedge \lambda_2 \geq 0 \wedge \lambda_1 + \lambda_2 = 1)$$

   defines the predicate **between**(p_1, p_2, p_3).
2. The real formula $(p_1^1 - p_2^1)^2 + (p_1^2 - p_2^2)^2 = (p_3^1 - p_4^1)^2 + (p_3^2 - p_4^2)^2$ defines the predicate **equidistance**(p_1, p_2, p_3, p_4).
3. The real formula $(p_1^1 - p_2^1)^2 + (p_1^2 - p_2^2)^2 = 1$ defines the predicate **unitdistance**(p_1, p_2).

[10]The predicates **between** and **equidistance** were introduced by Tarski [15].

[11]We use "hatted" symbols to denote point variables.

4. The real formula $(p_1^1 - o^1)(p_1^2 - o^2) - (p_2^1 - o^1)(p_2^2 - o^2) > 0$ defines the predicate $\mathbf{positive}(o, p_1, p_2)$.

5. The real formulas $p_1^1 \leq p_2^1$ and $p_1^2 \leq p_2^2$ define the predicates $\mathbf{smaller}_1(p_1, p_2)$ and $\mathbf{smaller}_2(p_1, p_2)$, respectively.

Example 5.1 Consider again the queries in Example 3.2.

Query 4 can be expressed in FO[**between**] as

$$\{\hat{v} \mid (\exists \hat{v}_1)(\exists \hat{v}_2)(\exists \hat{v}_3)(S(\hat{v}_1) \wedge S(\hat{v}_2) \wedge S(\hat{v}_3) \wedge \mathit{triangle}(\hat{v}, \hat{v}_1, \hat{v}_2, \hat{v}_3))\},$$

where $\mathit{triangle}(\hat{v}, \hat{v}_1, \hat{v}_2, \hat{v}_3)$ states of the point $\hat{v}$ that is in the triangle spanned by the points $\hat{v}_1$, $\hat{v}_2$, and $\hat{v}_3$. The predicate $\mathit{triangle}(\hat{v}, \hat{v}_1, \hat{v}_2, \hat{v}_3)$ is an abbreviation for the formula

$$(\exists \hat{p})(\mathbf{between}(\hat{v}_1, \hat{p}, \hat{v}_2) \wedge \mathbf{between}(\hat{v}_3, \hat{v}, \hat{p})).$$

Query 5 can be expressed in FO[**between**, **equidistance**] as

$$(\exists \hat{c})(\exists \hat{u})(\exists \hat{v})(\forall \hat{p})(R(\hat{p}) \Leftrightarrow \mathbf{equidistance}(\hat{c}, \hat{p}, \hat{u}, \hat{v})). \qquad \square$$

We intend to show that, for each of the sets Π of point predicates listed in Table 2, the language FO[Π] captures precisely all geometric queries expressible in FO[$\mathbf{R}$] that are generic with respect to the corresponding genericity notion.

Genericity notion	Point predicate set Π
Affine genericity	$\{\mathbf{between}\}$
Similarity genericity	$\{\mathbf{between}, \mathbf{equidistance}\}$
Isometry genericity	$\{\mathbf{between}, \mathbf{equidistance}, \mathbf{unitdistance}\}$
Direct-isometry genericity	$\{\mathbf{between}, \mathbf{equidistance}, \mathbf{unitdistance}, \mathbf{positive}\}$
Translation genericity	$\{\mathbf{between}, \mathbf{equidistance}, \mathbf{unitdistance}, \mathbf{positive}, \mathbf{smaller}_1, \ldots, \mathbf{smaller}_n\}$

Table 2: Point predicate sets for various geometric genericity notions.

We first consider the language FO[**between**]. We are going to show that FO[$\mathbf{R}$]-formulas can be simulated by FO[**between**] formulas that are parameterized by a basis, in a way that we shall make precise in Lemma 5.2.

It is well-known (e.g., [12]) that there exists a formula in the language (**between**) which defines the predicate $\mathit{basis}(\hat{z}_0, \hat{z}_1, \ldots, \hat{z}_n)$ which is *true* for the points $o, e_1, \ldots, e_n$ if $(o, e_1, \ldots, e_n)$ is a basis of $\mathbf{R}^n$.

In a basis $(o, e_1, \ldots, e_n)$, we associate to any real number ρ the point p on the line oe_1 for which $\overrightarrow{op} = \rho\, \overrightarrow{oe_1}$, i.e., the point on the first coordinate axis with coordinate ρ. Conversely, each point p on the line oe_1 is associated to the real number ρ for which $\overrightarrow{op} = \rho\, \overrightarrow{oe_1}$. We shall denote this real number ρ as $\overrightarrow{op}/\overrightarrow{oe_1}$.

It is also well-known (e.g., [12]) that the arithmetic operations on these numbers are first-order-expressible in the language (**between**). Therefore, we may assume the existence of the point predicates $less(\hat{z}_0, \hat{z}_1, \ldots, \hat{z}_n, \hat{x}, \hat{y})$, $plus(\hat{z}_0, \hat{z}_1, \ldots, \hat{z}_n, \hat{x}, \hat{y}, \hat{z})$, and $times(\hat{z}_0, \hat{z}_1, \ldots, \hat{z}_n, \hat{x}, \hat{y}, \hat{z})$, such that

- $less(o, e_1, \ldots, e_n, p, q)$ is *true* if $(o, e_1, \ldots, e_n)$ is a basis, p and q are points on the line oe_1, and $\overrightarrow{op}/\overrightarrow{oe_1} \leq \overrightarrow{oq}/\overrightarrow{oe_1}$;
- $plus(o, e_1, \ldots, e_n, p, q, r)$ is *true* if $(o, e_1, \ldots, e_n)$ is a basis, p, q, and r are points on the line oe_1, and $\overrightarrow{op}/\overrightarrow{oe_1} + \overrightarrow{oq}/\overrightarrow{oe_1} = \overrightarrow{or}/\overrightarrow{oe_1}$; and
- $times(o, e_1, \ldots, e_n, p, q, r)$ is *true* if $(o, e_1, \ldots, e_n)$ is a basis, p, q, and r are points on the line oe_1, and $\overrightarrow{op}/\overrightarrow{oe_1} \times \overrightarrow{oq}/\overrightarrow{oe_1} = \overrightarrow{or}/\overrightarrow{oe_1}$.

We also need the predicate $coordinates(\hat{z}_0, \hat{z}_1, \ldots, \hat{z}_n, \hat{u}, \hat{u}_1, \ldots, \hat{u}_n)$ that is *true* for the points $o, e_1, \ldots, e_n, p, p_1, \ldots, p_n$ if $(o, e_1, \ldots, e_n)$ is a basis of $\mathbf{R}^n$, $p_1, \ldots, p_n$ are points on the line oe_1, and

$$\overrightarrow{op} = \frac{\overrightarrow{op_1}}{\overrightarrow{oe_1}}\ \overrightarrow{oe_1} + \cdots + \frac{\overrightarrow{op_n}}{\overrightarrow{oe_1}}\ \overrightarrow{oe_n}.$$

In other words, one can think of the points p_1 through p_n on the line oe_1 as representing the coordinates of the point p. In subsequent results, the *coordinates* predicate will be used to associate points with their respective coordinates. As shown in [12] (Chapter 16, pages 163–164), the predicate *coordinates* can be defined by an FO[**between**] formula.

Finally, we observe that the predicate $collinear(\hat{x}, \hat{y}, \hat{z})$, which is *true* for the points p, q, and r if p, q, and r are collinear, can be expressed as

$$\mathbf{between}(\hat{x}, \hat{y}, \hat{z}) \lor \mathbf{between}(\hat{y}, \hat{x}, \hat{z}) \lor \mathbf{between}(\hat{z}, \hat{y}, \hat{x}).$$

Using the predicates introduced above, we can simulate FO[$\mathbf{R}$] formulas by FO[**between**] formulas that are parameterized by a basis, in the following sense.

Lemma 5.2 *Let σ be a relational schema. For each* FO[$\mathbf{R}$] *formula $\xi(x_1, \ldots, x_m)$ over $\overline{\sigma}$, there exists an* FO[**between**] *formula $\hat{\xi}(\hat{z}_0, \hat{z}_1, \ldots, \hat{z}_n, \hat{x}_1, \ldots, \hat{x}_m)$ over σ, with $\hat{z}_0, \hat{z}_1, \ldots, \hat{z}_n$, $\hat{x}_1, \ldots, \hat{x}_m$ free point variables, such that, for each geometric database $\mathcal{D}$ over σ in $\mathbf{R}^n$, for each basis $(o, e_1, \ldots, e_n)$ of $\mathbf{R}^n$, and for all points $p_1, \ldots, p_m$ on the line oe_1,*

$$\mathcal{D} \models \hat{\xi}(o, e_1, \ldots, e_n, p_1, \ldots p_m)$$

if and only if

$$\alpha(\overline{\mathcal{D}}) \models \xi\left(\frac{\overrightarrow{op_1}}{\overrightarrow{oe_1}}, \ldots, \frac{\overrightarrow{op_m}}{\overrightarrow{oe_1}}\right),$$

where α is the unique affinity of $\mathbf{R}^n$ mapping the basis $(o, e_1, \ldots, e_n)$ into the standard basis of $\mathbf{R}^n$.

Proof. Without loss of generality, we may assume that every atomic subformula of ξ is either of the form $x \leq y$, $x + y = z$, $x \times y = z$, $x = 0$, or $x = 1$, where x, y and z are real variables, or a relational atom in which only real variables occur. We now prove Lemma 5.2 by structural induction.

1. If ξ is $x \leq y$, then $\widehat{\xi}$ is $\mathit{less}(\widehat{z}_0, \widehat{z}_1, \ldots, \widehat{z}_n, \widehat{x}, \widehat{y})$.
2. If ξ is $x + y = z$, then $\widehat{\xi}$ is $\mathit{plus}(\widehat{z}_0, \widehat{z}_1, \ldots, \widehat{z}_n, \widehat{x}, \widehat{y}, \widehat{z})$.
3. If ξ is $x \times y = z$, then $\widehat{\xi}$ is $\mathit{times}(\widehat{z}_0, \widehat{z}_1, \ldots, \widehat{z}_n, \widehat{x}, \widehat{y}, \widehat{z})$.
4. If ξ is $x = 0$, then $\widehat{\xi}$ is $\widehat{x} = \widehat{z}_0$.
5. If ξ is $x = 1$, then $\widehat{\xi}$ is $\widehat{x} = \widehat{z}_1$.
6. If ξ is a relational atom $\overline{R}(x_1^1, \ldots, x_n^1, \ldots, x_1^m, \ldots, x_n^m)$, with R a relation name of arity m in σ, then

   $$\widehat{\xi} \equiv (\exists \widehat{u}_1) \ldots (\exists \widehat{u}_m)(R(\widehat{u}_1, \ldots, \widehat{u}_m) \wedge \bigwedge_{i=1}^{m} \mathit{coordinates}(\widehat{z}_0, \widehat{z}_1, \ldots, \widehat{z}_n, \widehat{u}_i, \widehat{x}_1^i, \ldots, \widehat{x}_n^i)).$$

   To see that this translation is correct, we need to make some observations about the *coordinates* predicate. Given a point p in $\mathbf{R}^n$ and given points $p_1, \ldots, p_n$ on the line oe_1, we know that $\mathit{coordinates}(o, e_1, \ldots, e_n, p, p_1, \ldots, p_n)$ is *true* if and only if

   $$\overrightarrow{op} = \frac{\overrightarrow{op_1}}{\overrightarrow{oe_1}} \, \overrightarrow{oe_1} + \cdots + \frac{\overrightarrow{op_n}}{\overrightarrow{oe_1}} \, \overrightarrow{oe_n}.$$

   Since affinities preserve linear combinations of vectors, the above equality is equivalent to

   $$\alpha(\overrightarrow{op}) = \frac{\overrightarrow{op_1}}{\overrightarrow{oe_1}} \, \alpha(\overrightarrow{oe_1}) + \cdots + \frac{\overrightarrow{op_n}}{\overrightarrow{oe_1}} \, \alpha(\overrightarrow{oe_n}).$$

   Since α maps the basis $(o, e_1, \ldots, e_n)$ to the standard basis of $\mathbf{R}^n$, the above equality is equivalent to

   $$\alpha(p) = \left(\frac{\overrightarrow{op_1}}{\overrightarrow{oe_1}}, \ldots, \frac{\overrightarrow{op_n}}{\overrightarrow{oe_1}} \right).$$

   Thus, given $p_1^1, \ldots, p_n^1, \ldots, p_1^m, \ldots, p_n^m$ on the line oe_1, we have that

   $$\mathcal{D} \models \widehat{\xi}(o, e_1, \ldots, e_n, p_1^1, \ldots, p_n^1, \ldots, p_1^m, \ldots, p_n^m)$$

   if and only if

   $$\alpha(\overline{\mathcal{D}}) \models \overline{R} \left(\frac{\overrightarrow{op_1^1}}{\overrightarrow{oe_1}}, \ldots, \frac{\overrightarrow{op_n^1}}{\overrightarrow{oe_1}}, \ldots, \frac{\overrightarrow{op_1^m}}{\overrightarrow{oe_1}}, \ldots, \frac{\overrightarrow{op_n^m}}{\overrightarrow{oe_1}} \right).$$

7. If ξ is $\neg\psi$, then $\widehat{\xi}$ is $\neg\widehat{\psi}$.

8. If ξ is $\psi \vee \vartheta$, then $\widehat{\xi}$ is $\widehat{\psi} \vee \widehat{\vartheta}$.

9. If ξ is $(\exists x)\psi(x, x_1, \ldots, x_m)$, then $\widehat{\xi}$ is

$$(\exists \widehat{x})(\mathit{collinear}(\widehat{z}_0, \widehat{z}_1, \widehat{x}) \wedge \widehat{\psi}(\widehat{z}_0, \widehat{z}_1, \ldots, \widehat{z}_n, \widehat{x}, \widehat{x}_1, \ldots, \widehat{x}_m)).$$

   To see that this translation is correct, we observe that, given points $p_1, \ldots, p_m$ on the line oe_1, $\mathcal{D} \models \widehat{\xi}(o, e_1, \ldots, e_n, p_1, \ldots, p_m)$ if and only if there exists a point p on the line oe_1 such that $\mathcal{D} \models \widehat{\psi}(o, e_1, \ldots, e_n, p, p_1, \ldots, p_m)$. By induction, this statement is equivalent to the existence of a point p on the line oe_1 such that

$$\alpha(\overline{\mathcal{D}}) \models \psi\left(\frac{\overrightarrow{op}}{\overrightarrow{oe_1}}, \frac{\overrightarrow{op_1}}{\overrightarrow{oe_1}}, \ldots, \frac{\overrightarrow{op_m}}{\overrightarrow{oe_1}}\right).$$

   Since each real number can be written as $\overrightarrow{op}/\overrightarrow{oe_1}$ for some point p on the line oe_1, the above statement is equivalent to

$$\alpha(\overline{\mathcal{D}}) \models \xi\left(\frac{\overrightarrow{op_1}}{\overrightarrow{oe_1}}, \ldots, \frac{\overrightarrow{op_m}}{\overrightarrow{oe_1}}\right).$$

This completes the proof of Lemma 5.2. $\square$

To show that FO[**R**]-expressible affine-generic queries are expressible in FO[**between**], we must somehow eliminate the bases which parameterize the formulas in FO[**between**] that simulate FO[**R**] formulas. We shall use affine genericity to show that these bases can be eliminated properly. To do so, we will first prove a genericity result that holds for geometric queries that are expressible by FO[**R**] formulas, and that is at the core of the elimination of bases.

Lemma 5.3 *Let G be a semi-algebraic group of transformations of $\mathbf{R}^n$. Let σ be a relational schema, and let $\mathcal{D}$ be a geometric database over σ in $\mathbf{R}^n$. Let $\overline{\mathcal{D}}$ be the underlying semi-algebraic database over $\overline{\sigma}$. Let ψ be an* FO[**R**] *formula expressing a G-generic geometric query over σ in $\mathbf{R}^n$. Then, for each g in G, $\psi(g(\overline{\mathcal{D}})) = g(\psi(\overline{\mathcal{D}}))$.*

Proof. Lemma 5.3 is non-trivial, because a semi-algebraic group of transformation may (and, in general, will) contain transformations whose coordinates are not all real algebraic. If g is such a transformation, $g(\overline{\mathcal{D}})$ need not be semi-algebraic, whence the definition of G-genericity cannot be applied to $\overline{\mathcal{D}}$, g, and ψ. However, if all coordinates of g are real algebraic, then $g(\overline{\mathcal{D}})$ is semi-algebraic, and $\psi(g(\overline{\mathcal{D}})) = g(\psi(\overline{\mathcal{D}}))$, by G-genericity. Hence, the following real sentence is *true* about the field of real algebraic numbers:

$$(\forall g \in G)(\psi(g(\overline{\mathcal{D}})) = g(\psi(\overline{\mathcal{D}})).$$

Since Tarski [14] showed that the field of the real algebraic numbers and the field of the real numbers are elementary equivalent, it follows that this sentence is also *true* about the field of the real numbers, whence the lemma holds. $\square$

We can now round off our investigation of the language FO[**between**].

Proposition 5.4 *The query language* FO[**between**] *expresses exactly all affine-generic geometric queries expressible in* FO[**R**].

Proof. First, we observe that queries expressed in FO[**between**] are indeed affine-generic, since FO[$\emptyset$] preserves arbitrary permutations of $\mathbf{R}^n$, and since the ternary betweenness relation on $\mathbf{R}^n$ is invariant under all affine transformations of $\mathbf{R}^n$.

Now, consider a k-ary affine-generic geometric query over the schema σ in $\mathbf{R}^n$, expressed by an FO[**R**] formula $\psi(x_1^1, \ldots, x_n^1, \ldots, x_1^k, \ldots, x_n^k)$ over $\overline{\sigma}$.

By Lemma 5.2, there exists an FO[**between**] formula

$$\widehat{\psi}(\widehat{z}_0, \widehat{z}_1, \ldots, \widehat{z}_n, \widehat{x}_1^1, \ldots, \widehat{x}_n^1, \ldots, \widehat{x}_1^k, \ldots, \widehat{x}_n^k)$$

over σ, with $\widehat{z}_0, \widehat{z}_1, \ldots, \widehat{z}_n$, $\widehat{x}_1^1, \ldots, \widehat{x}_n^1, \ldots, \widehat{x}_1^k, \ldots, \widehat{x}_n^k$ free point variables, such that, for each database $\mathcal{D}$ over σ, for each basis $(o, e_1, \ldots, e_n)$ of $\mathbf{R}^n$, and for all points $p_1^1, \ldots, p_n^1, \ldots, p_1^k, \ldots, p_n^k$ on the line oe_1,

$$\mathcal{D} \models \widehat{\psi}(o, e_1, \ldots, e_n, p_1^1, \ldots, p_n^1, \ldots, p_1^k, \ldots, p_n^k)$$

if and only if

$$\alpha(\overline{\mathcal{D}}) \models \psi\left(\frac{\overrightarrow{op_1^1}}{\overrightarrow{oe_1}}, \ldots, \frac{\overrightarrow{op_n^1}}{\overrightarrow{oe_1}}, \ldots, \frac{\overrightarrow{op_1^k}}{\overrightarrow{oe_1}}, \ldots, \frac{\overrightarrow{op_n^k}}{\overrightarrow{oe_1}}\right),$$

where α is the unique affinity of $\mathbf{R}^n$ mapping the basis $(o, e_1, \ldots, e_n)$ into the standard basis of $\mathbf{R}^n$.

Consider the following FO[**between**]-formulas φ' and φ:

$$\begin{aligned}
&\varphi'(\widehat{z}_0, \widehat{z}_1, \ldots, \widehat{z}_n, \widehat{x}_1^1, \ldots, \widehat{x}_n^1, \ldots, \widehat{x}_1^k, \ldots, \widehat{x}_n^k) \equiv \mathit{basis}(\widehat{z}_0, \widehat{z}_1, \ldots, \widehat{z}_n) \wedge \\
&\quad \bigwedge_{i=1}^{k} \bigwedge_{j=1}^{n} \mathit{collinear}(\widehat{z}_0, \widehat{z}_1, \widehat{x}_j^i) \wedge \widehat{\psi}(\widehat{z}_0, \widehat{z}_1, \ldots, \widehat{z}_n, \widehat{x}_1^1, \ldots, \widehat{x}_n^1, \ldots, \widehat{x}_1^k, \ldots, \widehat{x}_n^k); \text{ and} \\
&\varphi(\widehat{v}_1, \ldots, \widehat{v}_k) \equiv (\exists \widehat{z}_0)(\exists \widehat{z}_1) \ldots (\exists \widehat{z}_n)(\exists \widehat{x}_1^1) \ldots (\exists \widehat{x}_1^n) \ldots (\exists \widehat{x}_k^1) \ldots (\exists \widehat{x}_k^n) \\
&\qquad\qquad (\varphi'(\widehat{z}_0, \widehat{z}_1, \ldots, \widehat{z}_n, \widehat{x}_1^1, \ldots, \widehat{x}_1^n, \ldots, \widehat{x}_k^1, \ldots, \widehat{x}_k^n) \wedge \\
&\qquad\qquad \bigwedge_{i=1}^{k} \mathit{coordinates}(\widehat{z}_0, \widehat{z}_1, \ldots, \widehat{z}_n, \widehat{v}_i, \widehat{x}_i^1, \ldots, \widehat{x}_i^n)).
\end{aligned}$$

Formula φ expresses a k-ary geometric query over σ in $\mathbf{R}^n$.

To see the effect of this query, let $\mathcal{D}$ be a geometric database over σ in $\mathbf{R}^n$. Let $(o, e_1, \ldots, e_n)$ be an arbitrary basis of $\mathbf{R}^n$ and let α be the affinity mapping the basis $(o, e_1, \ldots, e_n)$ into the standard basis of $\mathbf{R}^n$. Consider the partial output of φ obtained by substituting $o, e_1, \ldots, e_n$ for $\widehat{z}_0, \widehat{z}_1, \ldots, \widehat{z}_n$, respectively.

It follows from Lemma 5.2 that φ simulates ψ (with points on the line oe_1 being used to represent real numbers), with the exception that the n components of a point in $\mathbf{R}^n$ are its coordinates with respect to the standard basis, whereas in φ, their representations refer to the basis $(o, e_1, \ldots, e_n)$. Thus, the partial output of φ considered equals $\alpha^{-1}(\psi(\alpha(\overline{\mathcal{D}})))$. Since there is a one-to-one correspondence between the affinities and the bases of $\mathbf{R}^n$, it follows that $\varphi(\mathcal{D}) = \bigcup_\alpha \alpha^{-1}(\psi(\alpha(\overline{\mathcal{D}})))$, where α ranges over all affinities of $\mathbf{R}^n$. Since ψ expresses an affine-generic geometric query, it follows from Lemma 5.3 that, for each affinity α of $\mathbf{R}^n$, $\alpha^{-1}(\psi(\alpha(\overline{\mathcal{D}}))) = \psi(\overline{\mathcal{D}})$, whence $\varphi(\mathcal{D}) = \psi(\overline{\mathcal{D}})$. □

This concludes the case of the language FO[**between**]. It now turns out that the other instances of Table 2 can be dealt with in almost the same way.

Theorem 5.5 *The query language* FO[Π] *expresses exactly all generic geometric queries expressible in* FO[**R**]*, with* Π *and the genericity type as listed in Table 2.*

Proof. The case where $\Pi = \{\mathbf{between}\}$ has been dealt with in Proposition 5.4. We next show that Theorem 5.5 holds for the other instances in Table 2.

A straightforward verification suffices to see that FO[Π] is sound relative to the FO[**R**]-expressible geometric queries of the corresponding genericity type.

The completeness proof is analogous to the proof of the completeness of FO[**between**] relative to the affine-generic queries. The only difference is that instead of working with arbitrary bases of $\mathbf{R}^n$, we need to work with bases appropriate for the genericity type considered. Thus we only need to know that there exists a formula in the language (Π) which characterizes these bases.

For the case that $\Pi = \{\mathbf{between}, \mathbf{equidistance}\}$, we need a formula in the language (**between**, **equidistance**) characterizing Euclidean bases. Such a formula is given in [12] (Definition 16.1, page 163). We denote this formula by $\mathit{basis}^{\mathit{Euclid}}$ for further use.

For the case that $\Pi = \{\mathbf{between}, \mathbf{equidistance}, \mathbf{unitdistance}\}$, we need a formula in the language (**between**, **equidistance**, **unitdistance**) that characterizes the Euclidean bases of unit length. The following is such a formula:

$$\mathit{basis}^{\mathit{Euclid}}(\widehat{z}_0, \widehat{z}_1, \ldots, \widehat{z}_n) \wedge \bigwedge_{i=1}^{n} \mathbf{unitdistance}(\widehat{z}_0, \widehat{z}_i).$$

We denote this formula by $\mathit{basis}^{\mathit{unit}}$ for further use.

For the case that $\Pi = \{\mathbf{between}, \mathbf{equidistance}, \mathbf{unitdistance}, \mathbf{positive}\}$, we need a formula in the language (**between**, **equidistance**, **unitdistance**, **positive**) characterizing the Euclidean bases of unit length which are oriented in the same way as the standard basis of $\mathbf{R}^n$. The following is such a formula:

$$\mathit{basis}^{\mathit{unit}}(\widehat{z}_0, \widehat{z}_1, \ldots, \widehat{z}_n) \wedge \mathbf{positive}(\widehat{z}_0, \widehat{z}_1, \ldots, \widehat{z}_n).$$

We denote this formula by $\mathit{basis}^{\mathit{positive}}$ for further use.

Finally, for the case that $\Pi = \{\mathbf{between}, \mathbf{equidistance}, \mathbf{unitdistance}, \mathbf{positive}, \mathbf{smaller}_1, \ldots, \mathbf{smaller}_n\}$, we need a formula in the language (**between**, **equidistance**, **unitdistance**, **positive**, $\mathbf{smaller}_1, \ldots, \mathbf{smaller}_n$) characterizing the bases which can be translated to the standard basis of $\mathbf{R}^n$. The following is such a formula:

$$\mathit{basis}^{\mathit{positive}}(\widehat{z}_0, \widehat{z}_1, \ldots, \widehat{z}_n) \wedge \bigwedge_{i=1}^{n} \mathbf{smaller}_i(\widehat{z}_0, \widehat{z}_i).$$

This completes the proof of Theorem 5.5. □

6 Complete geometric query languages

In Section 4.2, we showed that the language FO[$\mathbf{R}$] + while, when given appropriate semantics, is complete for various classes of geometric queries (see Theorem 4.6). While of interest, this result is unsatisfactory since FO[$\mathbf{R}$] + while does not have a natural geometric syntax. In this section, we augment the languages FO[Π] from the previous section with while-loops and show the more satisfactory result that the resulting languages FO[Π] + while, which *do* have a natural geometric syntax and semantics, are complete for the corresponding classes of geometric queries.

Let Π be a finite set of point predicates, and let σ be a relational schema. Syntactically, a *program* over σ in the query language FO[Π] + while is a finite sequence of *statements* and *while-loops*. Each statement has the form

$$R := \{(\widehat{v}_1, \ldots, \widehat{v}_k) \mid \varphi(\widehat{v}_1, \ldots, \widehat{v}_k)\},$$

with R a relation variable of arity k and φ a first-order formula in the language (Π) augmented with the relation names of σ and the previously introduced relation variables. Each while-loop has the form **while** φ **do** P, where P is a program and φ is a first-order sentence in the language (Π) augmented with the relation names of σ and the previously introduced relation variables.

Semantically, a program in the query language FO[Π] + while expresses a geometric query in the obvious way as soon as one of its relation variables has been designated as the output variable.

Theorem 6.1 *The query language* FO[Π]+while *expresses exactly all generic geometric queries, with* Π *and the genericity type as listed in Table 2.*

Proof. To simplify the exposition, we restrict ourselves to geometric queries in the plane, i.e., in $\mathbf{R}^2$. Furthermore, we will assume that $\sigma = \{R\}$ and that R is a unary relation. Finally, we only consider unary geometric queries, so the output is also a unary relation. (Such queries can be thought of as mapping point sets in the plane to points sets in the plane.) The proof we shall give can easily be generalized, however. Indeed, if we work in a higher-dimensional space, we only have to adjust each formula occurring in the proof to this case. If we have multiple input relations, of potentially different arities, we only have to encode each of them separately. (The encoding algorithm will need to consider the arity of an input relation.) Finally, if the output is k-ary, we only have to use an adapted version of the decoding algorithm described below.

We only develop the proof for the case where $\Pi = \{\textbf{between}\}$. For the other cases, it suffices to modify this proof as explained in the proof of Theorem 5.5.

It is clear that queries expressed in FO[**between**] + while are affine-generic.

We thus have to show that every unary geometric query Q over σ in $\mathbf{R}^2$ can be expressed by a program in FO[**between**] + while. The proof strategy we follow is that of Theorem 4.2, using insights gained from proving Theorem 4.6, and adopting techniques developed in the proof of Theorem 5.5. We first provide a sketch of this strategy:

1. *Encode*: Given a geometric database $\mathcal{D}$ over σ, we compute in FO[**between**]+while the natural number n such that $n = \mathrm{enc}(s)$, where s is the first string over Σ (defined as in the proof of Theorem 4.2, with $K = 2$) encoding the quantifier free FO[$\mathbf{R}$] formula defining $canon_G(\mathcal{D})$, where G is the group of affinities in the plane. We also compute $Type_G(\mathcal{D})$.

2. *Compute*: Let M be the counter machine that computes the query Q. We simulate in FO[**between**] + while the effect of running M on n.

3. *Decode*: In the case where the computation terminates with as output a natural number m that encodes a valid FO[$\mathbf{R}$] formula, we compute, again using an FO[**between**] + while program, its corresponding point set. This point set corresponds to $Q(canon_G(\mathcal{D}))$. Since Q is affine-generic, we have, for each g in $Type_G(\mathcal{D})$, that $Q(canon_G(\mathcal{D})) = Q(g(\mathcal{D})) = g(Q(\mathcal{D}))$. Therefore, to compute $Q(\mathcal{D})$, an FO[**between**] expression must be constructed which computes $\bigcup_{g \in Type_G(\mathcal{D})} g^{-1}(Q(canon_G(\mathcal{D})))$.

To accomplish this strategy, we need to realize that, unlike in FO[$\mathbf{R}$]+while, we have no direct access to real numbers in FO[**between**]+while. However, as should be clear from the techniques developed in the proofs of Theorem 5.5 and preceding auxiliary results, we can represent such real numbers relative to an arbitrary basis of the plane.

We now elaborate on each of the steps in our strategy.

1. *Encode*: The encoding program, shown in Figure 4, builds up relations T (for term) and F (for formula). The arity of T is $(n+1)+l+2 = 7$ (where $n = 2$, the dimension of the plane, and $l = 2$, n times the arity of R); each tuple in T is of the form $(o, e_1, e_2, t, p_1, p_2, \tau)$, where (o, e_1, e_2) is a basis of the plane, and t, p_1, p_2, and τ are points on the line oe_1 (of which we think as real numbers). More specifically, t is the encoding of a term which *only* uses the variables x_1 and x_2, and τ represents the value of t evaluated under the valuation $x_1 \mapsto p_1$ and $x_2 \mapsto p_2$. The arity of F is $(n+1)+l+1 = 6$. Each tuple in F is of the form $(o, e_1, e_2, f, p_1, p_2)$, where (o, e_1, e_2) is a basis of the plane, and f, p_1 and p_2 are points on the line oe_1 (of which we think as real numbers). More specifically, f is the encoding of a formula which *only* uses the variables x_1 and x_2, and $f(p_1, p_2)$ is *true*.

   In this program, the statement $n := 0$ is an abbreviation for the statement

   $$n := \{(\widehat{z}_0, \widehat{z}_1, \widehat{z}_2, \widehat{n}) \mid basis(\widehat{z}_0, \widehat{z}_1, \widehat{z}_2) \land \widehat{n} = \widehat{z}_0\},$$

   and the statement $n := n + 1$ is an abbreviation for the statement

   $$n := \{(\widehat{z}_0, \widehat{z}_1, \widehat{z}_2, \widehat{n}') \mid (\exists \widehat{n})(n(\widehat{z}_0, \widehat{z}_1, \widehat{z}_2, \widehat{n}) \land plus(\widehat{z}_0, \widehat{z}_1, \widehat{z}_2, \widehat{n}, \widehat{z}_1, \widehat{n}'))\}.$$

   The translations of the statements occurring under the various if-statements is straightforward. For example, the statement

   $$T := T \cup \{(o, e_1, e_2, n, p_1, p_2, p_1) \mid p_1, p_2 \in oe_1\}$$

```
$n := 0$; $T := \emptyset$; $F := \emptyset$;
$\mathit{Found} := \mathit{false}$;
**while** $\neg \mathit{Found}$ **do**
   $n := n + 1$;
   **if** $n$ encodes $x_1$ **then**
      $T := T \cup \{(o, e_1, e_2, n, p_1, p_2, p_1) \mid p_1, p_2 \in oe_1\}$ **else**
   **if** $n$ encodes $x_2$ **then**
      $T := T \cup \{(o, e_1, e_2, n, p_1, p_2, p_2) \mid p_1, p_2 \in oe_1\}$ **else**
   **if** $n$ encodes 0 **then**
      $T := T \cup \{(o, e_1, e_2, n, p_1, p_2, o) \mid p_1, p_2 \in oe_1\}$ **else**
   **if** $n$ encodes 1 **then**
      $T := T \cup \{(o, e_1, e_2, n, p_1, p_2, e_1) \mid p_1, p_2 \in oe_1\}$ **else**
   **if** $n$ encodes $(s + t)$ **then**
      $T := T \cup \{(o, e_1, e_2, n, p_1, p_2, c + d) \mid T(o, e_1, e_2, \mathrm{enc}(s), p_1, p_2, c) \wedge$
      $T(o, e_1, e_2, \mathrm{enc}(t), p_1, p_2, d)\}$ **else**
   **if** $n$ encodes $(s \times t)$ **then**
      $T := T \cup \{(o, e_1, e_2, n, p_1, p_2, cd) \mid T(o, e_1, e_2, \mathrm{enc}(s), p_1, p_2, c) \wedge$
      $T(o, e_1, e_2, \mathrm{enc}(t), p_1, p_2, d)\}$ **else**
   **if** $n$ encodes $(s \leq t)$ **then**
      $F := F \cup \{(o, e_1, e_2, n, p_1, p_2) \mid (\exists c)(\exists d)(T(o, e_1, e_2, \mathrm{enc}(s), p_1, p_2, c) \wedge$
      $T(o, e_1, e_2, \mathrm{enc}(t), p_1, p_2, d) \wedge c \leq d)\}$ **else**
   **if** $n$ encodes $(\neg\varphi)$ **then**
      $F := F \cup \{(o, e_1, e_2, n, p_1, p_2) \mid \neg F(o, e_1, e_2, \mathrm{enc}(\varphi), p_1, p_2)\}$ **else**
   **if** $n$ encodes $(\varphi \vee \psi)$ **then**
      $F := F \cup \{(o, e_1, e_2, n, p_1, p_2) \mid F(o, e_1, e_2, \mathrm{enc}(\varphi), p_1, p_2) \vee F(o, e_1, e_2, \mathrm{enc}(\psi), p_1, p_2)\}$;
   $\mathit{Found} := n$ encodes a formula which represents $\mathit{canon}_G(R)$;
**od**;
$n_{\mathit{canon}_G(R)} := n$;
$\mathit{Type}_G := \{g \in G \mid g(R) = \mathit{canon}_G(R)\}$.
```

Figure 4: The encoding program. Points on the line oe_1 are identified with real numbers.

is an abbreviation for the statement

$$
\begin{aligned}
T \;:=\; & \{(\widehat{z}_0,\widehat{z}_1,\widehat{z}_2,\widehat{m},\widehat{x},\widehat{y},\widehat{v}) \mid T(\widehat{z}_0,\widehat{z}_1,\widehat{z}_2,\widehat{m},\widehat{x},\widehat{y},\widehat{v}) \vee \\
& (n(\widehat{z}_0,\widehat{z}_1,\widehat{z}_2,\widehat{m}) \wedge \mathit{collinear}(\widehat{z}_0,\widehat{z}_1,\widehat{x}) \wedge \mathit{collinear}(\widehat{z}_0,\widehat{z}_1,\widehat{y}) \wedge \widehat{x}=\widehat{v})\}.
\end{aligned}
$$

In the statement

$$\mathit{Found} := n \text{ encodes a formula which represents } \mathit{canon}_G(R),$$

the part where we need to verify that the formula represents $\mathit{canon}_G(R)$ is an abbreviation for the sentence

$$
\begin{aligned}
& (\forall\widehat{z}_0)(\forall\widehat{z}_1)(\forall\widehat{z}_2)(\exists\widehat{a}_{11})(\exists\widehat{a}_{12})(\exists\widehat{a}_{21})(\exists\widehat{a}_{22})(\exists\widehat{b}_1)(\exists\widehat{b}_2)(\exists\widehat{m})(\mathit{basis}(z_0,z_1,z_2) \wedge \\
& \mathit{collinear}(\widehat{z}_0,\widehat{z}_1,\widehat{a}_{11}) \wedge \mathit{collinear}(\widehat{z}_0,\widehat{z}_1,\widehat{a}_{12}) \wedge \mathit{collinear}(\widehat{z}_0,\widehat{z}_1,\widehat{a}_{21}) \wedge \mathit{collinear}(\widehat{z}_0,\widehat{z}_1,\widehat{a}_{22}) \wedge \\
& \mathit{collinear}(\widehat{z}_0,\widehat{z}_1,\widehat{b}_1) \wedge \mathit{collinear}\,(\widehat{z}_0,\widehat{z}_1,\widehat{b}_2) \wedge \widehat{a}_{11}\widehat{a}_{22}-\widehat{a}_{12}\widehat{a}_{21}\neq 0 \wedge n(\widehat{z}_0,\widehat{z}_1,\widehat{z}_2,\widehat{m}) \wedge \\
& (\forall x')(\forall y')(F(\widehat{z}_0,\widehat{z}_1,\widehat{z}_2,\widehat{m},\widehat{x}',\widehat{y}') \Leftrightarrow (\exists\widehat{v})(\exists\widehat{x})(\exists\widehat{y})(R(\widehat{v}) \wedge \\
& \mathit{coordinates}(\widehat{z}_0,\widehat{z}_1,\widehat{v},\widehat{x},\widehat{y}) \wedge \widehat{x}'=\widehat{a}_{11}\widehat{x}+\widehat{a}_{12}\widehat{y}+\widehat{b}_1 \wedge \widehat{y}'=\widehat{a}_{21}\widehat{x}+\widehat{a}_{22}\widehat{y}+\widehat{b}_2))).
\end{aligned}
$$

Here, again, the subformulas $\widehat{a}_{11}\widehat{a}_{22}-\widehat{a}_{12}\widehat{a}_{21}\neq 0$, $\widehat{x}'=\widehat{a}_{11}\widehat{x}+\widehat{a}_{12}\widehat{y}+\widehat{b}_1$, and $\widehat{y}'=\widehat{a}_{21}\widehat{x}+\widehat{a}_{22}\widehat{y}+\widehat{b}_2$ can be seen as formulas in the language (**between**), expressed using the predicates *plus* and *times*.

Finally, the right-hand side of the statement

$$\mathit{Type}_G := \{g\in G \mid g(R)=\mathit{canon}_G(R)\}$$

is an abbreviation for

$$
\begin{aligned}
& \{(\widehat{z}_0,\widehat{z}_1,\widehat{z}_2,\widehat{a}_{11},\widehat{a}_{12},\widehat{a}_{21},\widehat{a}_{22},\widehat{b}_1,\widehat{b}_2) \mid (\exists\widehat{m})(\mathit{basis}(z_0,z_1,z_2) \wedge \\
& \mathit{collinear}(\widehat{z}_0,\widehat{z}_1,\widehat{a}_{11}) \wedge \mathit{collinear}(\widehat{z}_0,\widehat{z}_1,\widehat{a}_{12}) \wedge \mathit{collinear}(\widehat{z}_0,\widehat{z}_1,\widehat{a}_{21}) \wedge \mathit{collinear}(\widehat{z}_0,\widehat{z}_1,\widehat{a}_{22}) \wedge \\
& \mathit{collinear}(\widehat{z}_0,\widehat{z}_1,\widehat{b}_1) \wedge \mathit{collinear}\,(\widehat{z}_0,\widehat{z}_1,\widehat{b}_2) \wedge \widehat{a}_{11}\widehat{a}_{22}-\widehat{a}_{12}\widehat{a}_{21}\neq 0 \wedge n(\widehat{z}_0,\widehat{z}_1,\widehat{z}_2,\widehat{m}) \wedge \\
& (\forall x')(\forall y')(F(\widehat{z}_0,\widehat{z}_1,\widehat{z}_2,\widehat{m},\widehat{x}',\widehat{y}') \Leftrightarrow (\exists\widehat{v})(\exists\widehat{x})(\exists\widehat{y})(R(\widehat{v}) \wedge \\
& \mathit{coordinates}(\widehat{z}_0,\widehat{z}_1,\widehat{v},\widehat{x},\widehat{y}) \wedge \widehat{x}'=\widehat{a}_{11}\widehat{x}+\widehat{a}_{12}\widehat{y}+\widehat{b}_1 \wedge \widehat{y}'=\widehat{a}_{21}\widehat{x}+\widehat{a}_{22}\widehat{y}+\widehat{b}_2)))\}.
\end{aligned}
$$

A crucial aspect of this encoding program is that its while-loop terminates. The termination condition is determined by the last statement in the loop, i.e., the statement

$$\mathit{Found} := n \text{ encodes a formula which represents } \mathit{canon}_G(R).$$

We first observe that the relation n represents a unique natural number, in the sense that, if (o_1^1,e_1^1,e_2^1,n_1) and (o_1^2,e_1^2,e_2^2,n_2) are both in n, then (o_1^1,e_1^1,e_2^1) and (o_1^2,e_1^2,e_2^2) are both bases of the plane, and $\overrightarrow{o^1n_1}/\overrightarrow{o^1e_1^1}=\overrightarrow{o^2n_2}/\overrightarrow{o^2e_1^2}$.

Let $\mathcal{D}$ be the input to our query Q. We claim that the algorithm finds in $n_{\mathit{canon}_G(R)}$ the encoding of the formula that represents $\mathit{canon}_G(\mathcal{D})$ eventually, and thus sets *Found* to *true*.

To see this, consider the following property of the F relation. Let $\mathbf{b}=(o,e_1,e_2)$ be a basis of the plane and let $n_{\mathbf{b}}$ be the point on the line oe_1 representing the natural number n. Now consider the point set

$$F_{\mathbf{b}}^n = \{\widehat{v} \mid (\exists\widehat{x})(\exists\widehat{y})(F(o,e_1,e_2,n_{\mathbf{b}},\widehat{x},\widehat{y}) \wedge \mathit{coordinates}(o,e_1,e_2,\widehat{v},\widehat{x},\widehat{y}))\}.$$

Then, for each pair of bases $\mathbf{b}_1$ and $\mathbf{b}_2$ of the plane, and for each natural number n, $\gamma(F^n_{\mathbf{b}_1}) = F^n_{\mathbf{b}_2}$, where γ is the unique affine transformation mapping basis $\mathbf{b}_1$ to basis $\mathbf{b}_2$. This implies that, if there exists an affine transformation g such that $g(\mathcal{D}) = F^n_{\mathbf{b}_1}$, then there exists an affine transformation h such that $h(\mathcal{D}) = F^n_{\mathbf{b}_2}$, e.g., $h = \gamma \circ g$. This property entails that the while-loop terminates and that the program computes in $n_{canon_G(R)}$ the encoding of the formula that represents $canon_G(\mathcal{D})$.

2. *Compute*: In this phase, we simulate in FO[**between**] + while the counter machine M corresponding to the given query Q. The input to this program will be $n_{canon_G(R)}$. Let m be the output variable of this program. Either the program will diverge or else it will report its answer in m. We may assume, without loss of generality, that, if the program halts, the contents of m is a natural number representing a valid formula. In this case, this natural number necessarily corresponds to a formula representing the point set $Q(canon_G(\mathcal{D}))$.

3. *Decode*: We finally describe the program that decodes the result in m in the correct output of Q, i.e., the point set $Q(\mathcal{D})$. This program is the same as the encode program in Figure 4, except that the last line in the while-loop is replaced by the statement
$$Found := n = n_{canon_G(R)}.$$
Furthermore, the last two statements in the encode program are replaced by an assignment to the unary relation variable *Result* of the query
$$\{(\widehat{v}) \mid (\exists\widehat{z}_0)(\exists\widehat{z}_1)(\exists\widehat{z}_2)(\exists\widehat{a}_{11})(\exists\widehat{a}_{12})(\exists\widehat{a}_{21})(\exists\widehat{a}_{22})(\exists\widehat{b}_1)(\exists\widehat{b}_2)(\exists\widehat{m})(\exists\widehat{x})(\exists\widehat{y})(\exists\widehat{x}')(\exists\widehat{y}')$$
$$(\mathit{Type}_G(\widehat{z}_0,\widehat{z}_1,\widehat{z}_2,\widehat{a}_{11},\widehat{a}_{12},\widehat{a}_{21},\widehat{a}_{22},\widehat{b}_1,\widehat{b}_2) \wedge n(\widehat{z}_0,\widehat{z}_1,\widehat{z}_2,\widehat{m}) \wedge \mathit{coordinates}(\widehat{z}_0,\widehat{z}_1,\widehat{v},\widehat{x},\widehat{y}) \wedge$$
$$\widehat{x}' = \widehat{a}_{11}\widehat{x} + \widehat{a}_{12}\widehat{y} + \widehat{b}_1 \wedge \widehat{y}' = \widehat{a}_{21}\widehat{x} + \widehat{a}_{22}\widehat{y} + \widehat{b}_2 \wedge F(\widehat{z}_0,\widehat{z}_1,\widehat{z}_2,\widehat{m},\widehat{x}',\widehat{y}'))\}.$$

This completes the proof of Theorem 6.1. □

7 Extension to models with non-spatial data

In the model we have been using so far, a database can contain semi-algebraic sets only. Practical spatial database models support, in addition to purely spatial data, also non-spatial data without geometrical interpretation, such as the data stored in classical relational databases. For example, for a road, one typically does not only want to store its appearance on a map as a curve (a semi-algebraic set), but also its name or number. In this section, we briefly outline how our completeness results can be carried over to this setting.

It is not difficult to extend the semi-algebraic database model to incorporate non-spatial data [11]. Each relation name R of the schema then has a composite arity $[m, k]$: m is the non-spatial arity of R, and k is the spatial arity of R. In a semi-algebraic database $\mathcal{D}$, $R^{\mathcal{D}}$ then is a subset of $\mathbf{U}^m \times \mathbf{R}^k$, where $\mathbf{U}$ is the universe of non-spatial values, such that (*i*) $\pi_{1,\ldots,m}(R^{\mathcal{D}})$ is finite, and (*ii*), for each tuple $(v_1, \ldots, v_m)$ in $\pi_{1,\ldots,m}(R^{\mathcal{D}})$, the set $\{(a_1, \ldots, a_k) \mid (v_1, \ldots, v_m, a_1, \ldots, a_k) \in R^{\mathcal{D}}\}$ must be a semi-algebraic subset of $\mathbf{R}^k$. A representation of $R^{\mathcal{D}}$ is now no longer simply a real formula defining it, but a finite $(m+1)$-ary relation, where m is the non-spatial arity of R, consisting of tuples $(v_1, \ldots, v_m, \varphi)$, where $(v_1, \ldots, v_m)$ is in $\pi_{1,\ldots,m}(R^{\mathcal{D}})$ and φ is a real formula defining $\{(a_1, \ldots, a_k) \mid (v_1, \ldots, v_m, a_1, \ldots, a_k) \in R^{\mathcal{D}}\}$. It is now straightforward to also extend the geometric database model to incorporate non-spatial data.

These extended models fit neatly in the model for the language EQL described by Chandra and Harel [3]. This language is an extension of the well-known QL, a complete language for generic queries on classical relational databases. The extension supports the appearance of fully interpreted data values in relations. In our application of this model, these interpreted data values are real formulas.

The key construct of EQL is an operator for going from an i-ary relation to the i-th interpreted data value. In a direct combination of the languages QL and FO[$\mathbf{R}$] + while, this construct can be expressed. The QL component of the combined language deals with the projection of the relations on the ordinary data columns, and the FO[$\mathbf{R}$] + while component deals with the spatial projection.

Based on this observation, it can be verified that the combined language, QL $\oplus$ (FO[$\mathbf{R}$] + while), expresses exactly all queries on semi-algebraic databases extended with non-spatial data. Similarly, it can be shown that the combined languages QL $\oplus$ (FO[Π] + while) express exactly all generic queries on geometric databases extended with non-spatial data, where Π and the genericity type is as listed in Table 2.

Acknowledgments

The authors wish to thank Jan Paredaens and Bart Kuijpers for interesting discussions related to generic geometric queries. We are grateful to the two anonymous referees whose feedback led to improvements in the presentation of the paper. The authors also wish to acknowledge their collaboration with Luc Vandeurzen to establish Corollary 4.3.

References

[1] S. Abiteboul and V. Vianu, "Procedural Languages for Database Queries and Updates," *Journal of Computer and System Sciences*, 41:2, 1990, pp. 181–229.

[2] M. Benedikt, G. Dong, L. Libkin, and L. Wong, "Relational Expressive Power of Constraint Query Languages," *Journal of the ACM*, 45:1, 1998, pp. 1–34.

[3] A. Chandra and D. Harel, "Computable Queries for Relational Data Bases," *Journal of Computer and System Sciences*, 21:2, 1980, pp. 156–178.

[4] H.B. Enderton, *A Mathematical Introduction to Logic*, Academic Press, New York, 1972.

[5] D. Gans, *Transformations and Geometries*, Meredith Corporation, New York, 1969.

[6] S. Grumbach and J. Su, "Queries with arithmetical constraints," *Theoretical Computer Science*, 173:1, 1997, pp. 151–181.

[7] 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.

[8] B. Kuijpers, J. Paredaens, and D. Suciu, unpublished results, University of Antwerp, 1995.

[9] G. McCarty, *Topology: An Introduction with Applications to Topological Groups*, McGraw-Hill Inc., 1967.

[10] C.H. Papadimitriou, D. Suciu, and V. Vianu, "Topological Queries in Spatial Databases," in Proceedings *15th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* (Montreal, Quebec, Canada), ACM Press, New York, 1996, pp. 81–92.

[11] J. Paredaens, J. Van den Bussche, and D. Van Gucht, "Towards a theory of spatial database queries," in Proceedings *13th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* (Minneapolis, Minnesota), ACM Press, New York, 1994, pp. 279–288.

[12] W. Schwabhäuser, W. Szmielew, and A. Tarski, *Metamathematische Methoden in der Geometrie*, Springer-Verlag, Berlin, 1983.

[13] J. Schwartz and M. Sharir, "On the 'Piano Movers' Problem. II. General Techniques for Computing Topological Properties of Real Algebraic Manifolds," *Advances in Applied Mathematics*, 4, 1983, pp. 298–351.

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

[15] A. Tarski, "What is Elementary Geometry?", in *The Axiomatic Method*, L. Henkin, P. Suppes, and A. Tarski, eds., North Holland Publishing Company, Amsterdam, 1959, pp. 16–29.

[16] L. Van Den Dries, "Alfred Tarski's Elimination Theory for Real Closed Fields," *Journal of Symbolic Logic*, 53, 1988, pp. 7–19.