First-Order Languages Expressing Constructible Spatial Database Queries

Bart Kuijpers*
University of Limburg (tUL/LUC)
Department WNI
B-3590 Diepenbeek, Belgium
bart.kuijpers@luc.ac.be

Gabriel Kuper
Università di Trento,
Dipartimento informatica
e telecommunicazioni
38050 Povo, Trento, Italia
kuper@acm.org

Jan Paredaens
University of Antwerp (UIA)
Dept. Math. & Computer Sci.,
Universiteitsplein 1,
B-2610 Antwerp, Belgium
pareda@uia.ac.be

Luc Vandeurzen
University of Limburg (tUL/LUC)
Department WNI
B-3590 Diepenbeek, Belgium
luc.vandeurzen@luc.ac.be

Abstract

The research presented in this paper is situated in the framework of constraint databases that was introduced by Kanellakis, Kuper, and Revesz in their seminal paper of 1990. Constraint databases and query languages are defined using real polynomial constraints. As a consequence of a classical result by Tarski, first-order queries in the constraint database model are effectively computable, and their result is again within the constraint model.

For reasons of efficiency, this model is implemented with only *linear* polynomial constraints. In this case, we also have a closure property: linear queries evaluated on linear databases yield linear databases. However, the limitation to linear polynomial constraints has severe implications on the expressive power of the query language. Indeed, the constraint database model has its most important applications in the field of spatial databases and, with only linear polynomials, the data modeling capabilities are limited and queries important for spatial applications that involve Euclidean distance are not expressible.

The aim of this paper is to identify a class of two-dimensional constraint databases and a query language within the constraint model that go beyond the linear model. Furthermore, this language should allow the expression of queries concerning distance. Hereto, we seek inspiration in the Euclidean constructions, i.e., constructions by ruler and compass. In the course of reaching our goal, we have studied three languages for ruler-and-compass constructions.

First, we present a programming language. We show that this programming language captures exactly the first-order ruler-and-compass constructions

*Contact author.

that are also expressible in the first-order constraint language with arbitrary polynomial constraints. If our programming language is extended with a **while** operator, we obtain a language that is complete for all ruler-and-compass constructions in the plane, using techniques of Engeler.

Secondly, we transform this programming language into a query language for finite point databases. We show that, on finite point databases, the full expressive power of this query language is that of the first-order language with arbitrary polynomial constraints. It is therefore too powerful for our purposes. We then consider a safe fragment of this language. Safe queries have the property that they map finite point relations to finite point relations that are constructible from the former using ruler and compass.

This safe fragment is the key ingredient in the formulation of our main result. We prove a closure property for the class of constraint databases consisting of those planar figures that can be described by means of linear constraints and those quadratic polynomials that describe circles. We call this class of databases the semi-circular databases. We define a query language on the class of semi-circular databases based on the notion of safe queries. When we restrict our attention to finite databases, this language captures exactly the ruler-and-compass constructions.

Finally, we compare the expressive power of this new query language with the expressive power of the first-order language with linear constraints on linear databases on the one hand and with the expressive power of the first-order language with polynomial constraints on semi-circular databases on the other hand.

1 Introduction and motivation

Kanellakis, Kuper, and Revesz [27] introduced the framework of *constraint databases* which provides a rather general model for spatial databases [29] (for an overview of results in the area of constraint databases we refer to a recent survey book [30]). Spatial database systems [1, 6, 10, 21, 22, 34] are concerned with the representation and manipulation of data that have a geometrical or topological interpretation. In the context of the constraint model, a spatial database, although conceptually viewed as a possibly infinite set of points in the real space, is represented as a finite union of systems of polynomial equations and inequalities. For example, the spatial database consisting of the set of points on the northern hemisphere together with the points on the equator of the unit sphere in the three-dimensional space $\mathbf{R}^3$ can be represented by the formula $x^2 + y^2 + z^2 = 1 \wedge z \geq 0$. The set $\{(x, y) \mid (y - x^2)(x^2 - y + 1/2) > 0\}$ of points in the real plane lying strictly above the parabola $y = x^2$ and strictly below the parabola $y = x^2 + 1/2$ is an example of a two-dimensional database in the constraint model. In mathematical terms, these sets are called *semi-algebraic*. For an overview of their properties, we refer to [5].

Several languages to query databases in the constraint model have been proposed and studied. A simple query language is obtained by extending the relational calculus with polynomial inequalities [29]. This language is usually referred to as $\mathsf{FO} + \mathsf{poly}$. The query deciding whether the two-dimensional database S is a straight line, for

instance, can be expressed in this language by the sentence

$$(\exists a)(\exists b)(\exists c)(\neg(a = 0 \wedge b = 0) \wedge ((\forall x)(\forall y)(S(x, y) \leftrightarrow ax + by + c = 0))).$$

Although variables in such expressions range over the real numbers, queries expressed in this calculus can still be computed effectively, and we have the closure property that says that an $\mathsf{FO} + \mathsf{poly}$ query, when evaluated on a spatial database in the constraint model, yields a spatial database in the constraint model. These properties are direct consequences of the quantifier elimination procedure for the first-order theory of real closed fields that was given by Tarski [35].

This quantifier elimination procedure is computationally very expensive, however (for an algorithmic description of quantifier elimination and an analysis of its complexity, we refer to [7, 8, 32]). The implementation of a query evaluation procedure based on quantifier elimination, for which the best algorithm is exponential in the number of quantified variables that are to be eliminated, therefore seems infeasible for real spatial database applications (we discuss this further below). In existing implementations of the constraint model (see the work of Grumbach et al. [15, 16, 18]), the constraints are restricted to *linear* polynomial constraints. The sets definable in this restricted model are called *semi-linear*. It is argued that linear polynomial constraints provide a sufficiently general framework for spatial database applications [18, 36]. Indeed, in geographical information systems, which is one of the main application areas for spatial databases, linear approximations are used to model geometrical objects (for an overview of this field since the early 90's we refer to [1, 6, 10, 21, 22, 34]).

When we extend the relational calculus with linear polynomial inequalities, we obtain an effective language with the closure property as above, but this time with respect to linear databases. We refer to this language as $\mathsf{FO} + \mathsf{lin}$, and, so we have the property that an $\mathsf{FO} + \mathsf{lin}$ query evaluated on a linear constraint database yields a linear constraint database.

We return to the complexity of query evaluation by quantifier elimination. Although both for $\mathsf{FO} + \mathsf{poly}$ and $\mathsf{FO} + \mathsf{lin}$ the cost of quantifier elimination grows exponentially with the number of quantified variables that have to be eliminated, an argument in favor of the language $\mathsf{FO} + \mathsf{lin}$ is that there exists a conceptually "easier" way to eliminate quantifiers for this language which makes an effective implementation feasible (see e.g. [15, 16, 18]). This easier algorithm is usually referred to as Fourier's method (see, e.g., [26, 28]). As mentioned, it is not more efficient than the elimination procedure for $\mathsf{FO} + \mathsf{poly}$. There is however a slight gain in data complexity: Grumbach et al. have shown that the data complexity for $\mathsf{FO} + \mathsf{lin}$ is NC^1 while it is NC for $\mathsf{FO} + \mathsf{poly}$ [19]. Another, more significant, advantage of the linear model is the existence of numerous efficient algorithms for geometrical operations [31].

These complexity issues are not our main concern, however. There are a number of serious disadvantages to the restriction to linear polynomial constraints, which mainly concern the limited expressive power of the query language $\mathsf{FO} + \mathsf{lin}$. The expressive power of the language $\mathsf{FO} + \mathsf{lin}$ has been extensively studied (see, e.g., [2, 3, 17, 20, 36, 37] and references therein). Among the limitations of $\mathsf{FO} + \mathsf{lin}$, one

of the most important is that the language is incapable of expressing queries that involve Euclidean distance, betweenness and collinearity. A query like "Return all cities in Belgium that are further than 100 km away from Brussels" is, however, a query that is of importance in spatial database applications.

The aim of this paper is to overcome these limitations of the linear model and its query language $\mathsf{FO} + \mathsf{lin}$ for two-dimensional spatial databases in the constraint model.

We note that languages whose expressive power on semi-linear databases is strictly between that of $\mathsf{FO} + \mathsf{lin}$ and $\mathsf{FO} + \mathsf{poly}$ have already been studied. Gyssens, Vandeurzen, and Van Gucht [36, 37] have shown that, even though $\mathsf{FO} + \mathsf{lin}$ extended with a primitive for collinearity yields a language with the complete expressive power of $\mathsf{FO} + \mathsf{poly}$, a "careful" extension with a collinearity operator yields a language whose expressive power is strictly between that of $\mathsf{FO} + \mathsf{lin}$ and that of $\mathsf{FO} + \mathsf{poly}$ on semi-linear databases. But even this extension does not allow the expression of queries involving distance, however.

We will define a query language $\mathsf{SafeEuQL}^{\uparrow}$ and a class of two-dimensional constraint databases on which this language is closed. We call these databases *semi-circular*. The language $\mathsf{SafeEuQL}^{\uparrow}$ allows the expression of queries concerning distance, betweenness, and collinearity. The class of semi-circular databases obviously is a strict superclass of the class of linear databases, since $\mathsf{SafeEuQL}^{\uparrow}$ allows for the definition of data by means of distance. Semi-circular databases are describable by means of polynomial equalities and inequalities that involve linear polynomials and polynomials that define circles. The language $\mathsf{SafeEuQL}^{\uparrow}$ is strictly more powerful on linear databases than $\mathsf{FO} + \mathsf{lin}$ and on semi-circular databases strictly less powerful than $\mathsf{FO} + \mathsf{poly}$.

The main goal of this paper is to define and describe a query language $\mathsf{SafeEuQL}^{\uparrow}$ with the above stated properties.

To accomplish this goal, we have turned, for inspiration, to the *Euclidean constructions*, i.e., the constructions by ruler and compass that we know from high-school geometry. These constructions were first described in the 4th century B.C. by Euclid of Alexandria in the thirteen books of his *Elements* [13, 24]. Of the 465 propositions to be found in these volumes only 60 are concerned with ruler-and-compass constructions. Most of these constructions belong to the mathematical folklore and are known to all of us. "Construct the perpendicular from a given point on a given line" or "construct a regular pentagon" are well-known examples. Since the 19th century, we also know that a certain number of constructions are *not* performable by ruler and compass, e.g., the trisection of an arbitrary angle or the squaring of the circle are impossible. For a 20th century description of these constructions and of the main results concerning them, we refer to [25].

In the course of reaching our goal, we have defined and studied three languages for ruler-and-compass constructions.

Firstly, we introduce a programming language that describes Euclidean constructions. We will refer to this procedural language as EuPL (short for Euclidean Programming Language). Engeler [11, 12] has studied a similar programming language in the 60's. Engeler's language, however, contains a while-loop and therefore

goes beyond first-order logic-based languages. His language also differs from our language in that EuPL also contains a choice-statement. This statement corresponds to choosing arbitrary points, that satisfy some conditions, in the plane, something that is often done in constructions with ruler and compass on paper. We claim that EuPL captures exactly the planar geometrical constructions, i.e., the first-order expressible ruler-and-compass constructions. We show that the choice-statement, at least for deterministic programs, can be omitted. We also prove a number of useful decidability properties of EuPL programs: equivalence and satisfiability of EuPL programs are decidable; it is decidable whether a program is deterministic.

Secondly, we transform the programming language EuPL into a query language for finite point databases, called EuQL (short for Euclidean Query Language). This calculus can express non-constructible queries. In fact, we show that the expressive power of EuQL is the same as that of FO + poly on finite point databases. It is therefore too powerful for our purposes.

We then study a *safe* fragment of EuQL, in which all queries are constructible. In particular, a SafeEuQL query returns constructible finite point relations from given finite point relations.

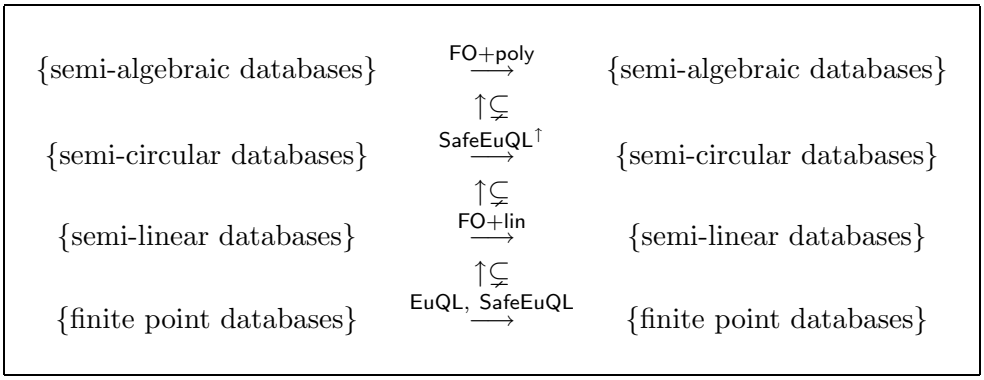
SafeEuQL is the key ingredient in our query language SafeEuQL$^{\uparrow}$ for semi-circular databases. Since SafeEuQL works on finite point databases, we interpret these queries to work on intensional representations of semi-circular databases. We then give FO + poly-definable mappings from one representation to the other. Using these mappings, we can "lift" SafeEuQL to a query language on semi-circular databases. This "lifting"-technique has also been used by Benedikt and Libkin [4] and Vandeurzen, Gyssens and Van Gucht [23].

We compare the expressive power of SafeEuQL$^{\uparrow}$ with the expressive power of FO + poly on semi-circular databases. Next, we show that on semi-linear databases FO + poly is more expressive than SafeEuQL$^{\uparrow}$ (the trisection of an angle is an example of a query that can be expressed in FO + poly but not in SafeEuQL$^{\uparrow}$). Finally, we compare the expressive power of SafeEuQL$^{\uparrow}$ and FO + lin on semi-linear databases.

Overview of the query languages. The following scheme gives an overview of the different languages. Horizontal arrow simply indicate that a language is closed on certain classes of databases. A subscheme of the form

$$\begin{array}{cc} & \xrightarrow{\mathcal{L}_1} \\ & \uparrow\subsetneq \\ A & \xrightarrow{\mathcal{L}_2} \end{array}$$

means that on databases in the class A, the language $\mathcal{L}_1$ is more expressive than the language $\mathcal{L}_2$.



Remark that the second horizontal arrow actually means that $\mathsf{SafeEuQL}^{\uparrow}$ works on finite point-representations of semi-circular databases and that it is closed on this class of objects.

Organization of the paper. In the next section, we define $\mathsf{FO} + \mathsf{poly}$ and $\mathsf{FO} + \mathsf{lin}$ as data modeling tools and query languages. In Section 3, we introduce the class of the semi-circular databases and describe a complete and lossless representation of them by means of finite point databases. We devote the next three sections to the study of the three languages for ruler-and-compass constructions: EuPL, EuQL, and SafeEuQL. The query language for semi-circular databases is given in Section 7, where we show that it is closed, and compare its expressive power with those of $\mathsf{FO} + \mathsf{lin}$ on semi-linear databases and $\mathsf{FO} + \mathsf{poly}$ on semi-circular databases.

2 Constraint-based database models

In this section, we provide the necessary background for the polynomial and linear constraint database models. We explain the notions of query in the context of these database models, and formally define two natural query languages, $\mathsf{FO} + \mathsf{poly}$ and $\mathsf{FO} + \mathsf{lin}$, for the polynomial and the linear database model, respectively. Since the linear database model is a sub-model of the polynomial database model, we start with the latter.

We denote the set of the real numbers by $\mathbf{R}$.

2.1 The polynomial database model

First, we define a *polynomial formula* as a well-formed first-order logic formula in the theory of the real numbers (i.e., over $(+, \times, <, 0, 1)$). In other words, a polynomial formula is built with the logical connectives $\wedge, \vee$, and $\neg$ and the quantifiers $\exists$ and $\forall$ from atomic formulas of the form $p(x_1, \ldots, x_n) > 0$, where $p(x_1, \ldots, x_n)$ is a polynomial with real algebraic coefficients and real variables $x_1, \ldots, x_n$.

Every polynomial formula $\varphi(x_1, \ldots, x_n)$ with n free variables $x_1, \ldots, x_n$ defines a point set

$$\{(x_1, \ldots, x_n) \in \mathbf{R}^n \mid \varphi(x_1, \ldots, x_n)\}$$

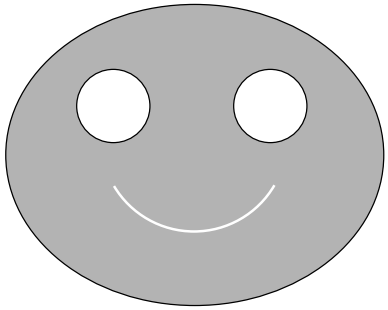


Figure 1: Example of a semi-algebraic relation.

in the n-dimensional Euclidean space $\mathbf{R}^n$ in the standard manner. Point sets defined by a polynomial formula are called *semi-algebraic sets*. We will also refer to them as *semi-algebraic relations*, since they can be seen as n-ary relations over the real numbers.

We remark that, by the quantifier-elimination theorem of Tarski [35], it is always possible to represent a semi-algebraic set by a quantifier-free formula. The same theorem also guarantees the decidability of the equivalence of two polynomial formulas.

A *polynomial database* is a finite set of *semi-algebraic relations*. Relations and databases can be represented finitely in this model by means of the corresponding polynomial formulas.

Example 2.1 Figure 1 shows an example of a binary semi-algebraic relation which is represented by the formula

$$x^2/25 + y^2/16 \leq 1 \wedge x^2 + 4x + y^2 - 2y \geq -4 \\ \wedge\, x^2 - 4x + y^2 - 2y \geq -4 \wedge (x^2 + y^2 - 2y \neq 8 \vee y > -1).$$

□

A query in the polynomial database model is defined as a mapping of m-tuples of semi-algebraic relations to a semi-algebraic relation, which

(i) must be computable; and,

(ii) must satisfy certain genericity conditions which are discussed at length by Paredaens, Van den Bussche, and Van Gucht [29].

The most natural query language for the polynomial data model is the relational calculus augmented with polynomial equalities and inequalities, i.e., the first-order language which contains as atomic formulas polynomial inequalities and formulas of

the form $R_i(y_1, \ldots, y_n)$, where R_i $(i = 1, \ldots, m)$ are semi-algebraic relation names for the input parameters of the query, and $y_1, \ldots, y_n$ are real variables. In the literature, this query language is commonly referred to as $\mathsf{FO} + \mathsf{poly}$ [30].

Example 2.2 The $\mathsf{FO} + \mathsf{poly}$ formula

$$R(x, y) \wedge (\forall \varepsilon)(\varepsilon \leq 0 \vee (\exists v)(\exists w)(\neg R(v, w) \wedge (x - v)^2 + (y - w)^2 < \varepsilon))$$

has x and y as free variables. For a given binary semi-algebraic relation R, it computes the set of points with coordinates (x, y) that belong to the intersection of R and its topological border. □

Tarski's quantifier-elimination procedure ensures that every $\mathsf{FO} + \mathsf{poly}$ query is effectively computable and yields a polynomial database as result [27].

2.2 The linear database model

Polynomial formulas built from atomic formulas that contain only linear polynomials with real algebraic coefficients are called *linear formulas*. Point sets defined by linear formulas are called *semi-linear sets* or *semi-linear relations*.

We remark that there is also a quantifier-elimination property for the linear model: any linear formula that contains quantifiers, can be converted to an equivalent quantifier-free linear formula. There is a conceptually easy algorithm, usually referred to as Fourier's method, to eliminate quantifiers in the linear model (this method is described in [26, 28]).

A *linear database* is a finite set of *semi-linear relations*. Relations and databases can be represented finitely in this model by means of the corresponding linear formulas.

Example 2.3 The binary semi-linear relation depicted in Figure 2 is described by the formula

$$\begin{aligned} x - 3 \leq 0 \quad & \wedge \quad x + 3 \geq 0 \wedge y - 3 \leq 0 \wedge y + 3 \geq 0 \\ & \wedge \quad \neg(x - 2 \leq 0 \wedge x - 1 \geq 0 \wedge y - 2 \leq 0 \wedge y - 1 \geq 0) \\ & \wedge \quad \neg(x + 1 \leq 0 \wedge x + 2 \geq 0 \wedge y - 2 \leq 0 \wedge y - 1 \geq 0) \\ & \wedge \quad \neg(x \geq -2 \wedge x \leq 0 \wedge 2y + x + 4 = 0) \\ & \wedge \quad \neg(x \geq 0 \wedge x \leq 2 \wedge y - x + 4 = 0). \end{aligned}$$

□

As in the polynomial model, queries in the linear model are defined as mappings from m-tuples of semi-linear relations to a semi-linear relation. A very appealing query language for the semi-linear data model, called $\mathsf{FO} + \mathsf{lin}$, is obtained by restricting the polynomial formulas in $\mathsf{FO} + \mathsf{poly}$ to contain only linear polynomials. Using algebraic computation techniques for the elimination of variables, one can see that the result of every $\mathsf{FO} + \mathsf{lin}$ query is a semi-linear relation [26, 28, 30].

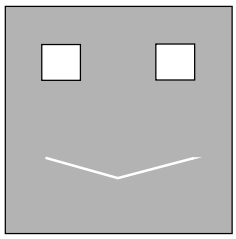


Figure 2: Example of a semi-linear relation.

Example 2.4 The $\mathsf{FO} + \mathsf{lin}$ formula

$$R(x, y) \wedge (\forall \varepsilon)(\varepsilon \leq 0 \vee (\exists v)(\exists w)(\neg R(v, w) \wedge x - \varepsilon < v < x + \varepsilon \wedge y - \varepsilon < w < y + \varepsilon))$$

has two free variables: x and y. For a given binary semi-linear relation R, it computes the set of points with coordinates (x, y) that belong to the intersection of R and its topological border. In fact, this formula is equivalent to the one in Example 2.2, even though it makes use of a different metric to compute the topological border. It should be clear that not every $\mathsf{FO} + \mathsf{poly}$ formula has an equivalent $\mathsf{FO} + \mathsf{lin}$ formula. □

3 Semi-circular relations

We now describe a class of planar relations in the constraint model that can be described by linear polynomials and those quadratic polynomials that describe circles. We also give a way to encode these relations as finite relations of points. This encoding is *complete* and *lossless*. This representation will be used later on.

Definition 3.1 We call a subset of $\mathbf{R}^2$ an *semi-circular set* or *semi-circular relation* if and only if it can be defined as a Boolean combination of sets of the form

$$\{(x, y) \mid ax + by + c\ \theta\ 0\},$$

or

$$\{(x, y) \mid (x - a)^2 + (y - b)^2\ \theta\ c^2\},$$

with a, b, and c real algebraic numbers, and $\theta \in \{\geq, >\}$. □

As far as planar figures are concerned, the class of semi-circular relations clearly contains the class of semi-linear relations.

Example 3.1 Figure 3 (a) shows an example of a semi-circular relation. It is the set

$$\{(x, y) \mid x^2 + y^2 \leq 1 \vee (y = 0 \wedge 1 \leq x < 2) \vee (x > 2 \wedge \neg y = 0)\}.$$

□

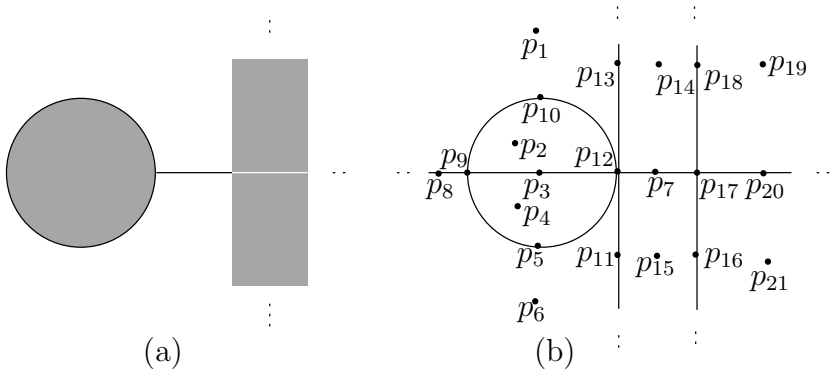


Figure 3: A semi-circular relation (a) and its carrier (b).

Given such an semi-circular database, we can then consider the figure consisting of all sets $\{(x, y) \mid p(x, y) = 0\}$ for each polynomial $p(x, y)$ in the definition of the semi-circular relation.

For the semi-circular relation of Figure 3 (a), these sets are shown in part (b) of Figure 3 and they are defined by the equations $x^2 + y^2 - 1 = 0$, $y = 0$, $x - 1 = 0$, and $x - 2 = 0$. We refer to these lines and circles as *a carrier* of the semi-circular relation (or as *the carrier* of a particular representation of the semi-circular relation). The carrier in Figure 3 (b) partitions the plane $\mathbf{R}^2$ into 21 classes, each of which belongs entirely to the semi-circular relation or to its complement. In general, these partition classes can be disconnected. Representatives $p_1, \ldots, p_{21}$ of each of these classes are shown in Figure 3 (b). We can therefore represent a semi-circular relation R by a finite point database[1] that consists of three relations, L, P and C, as follows:

- L contains, for each line in the carrier of R, a pair of its points;
- C contains, for each circle in the carrier of R, its center and one of its points;
- P contains a representative of each class in the partition induced by the carrier of R, that belongs to R.

L and C are binary relations of points in the plane. P is a unary relation of points.

We will refer to such a finite representation of a semi-circular relation as an *intensional LPC-representation*. Clearly, a semi-circular relation can have more than one intensional LPC-representation (in particular because it can have more than one constraint formula describing it).

For the semi-circular database of Figure 3 (a) we get the following finite representation: L consists of the tuples (p_7, p_8), (p_{11}, p_{12}), and (p_{16}, p_{17}); C consists of one single tuple (p_3, p_5); and P consists of the points p_9, p_{10}, p_2, p_3, p_4, p_5, p_{12}, p_7, p_{19}, and p_{21}.

[1]These points can be represented explicitly by their real algebraic coordinates, or implicitly by a real polynomial formula. Equality of such points can therefore be decided by Tarski's theorem [35].

We remark that an intensional LPC-representation of a semi-circular relation is lossless in the sense that it contains enough information to reconstruct the semi-circular relation it represents. In Section 7, we show how to compute in $\mathsf{FO} + \mathsf{poly}$ the LPC-representation of a semi-circular relation.

Finally, we give the following property.

Proposition 3.1 *It is decidable whether two LPC-representations of semi-circular relations represent the same semi-circular relation.*

Proof 3.1 Given an LPC-representation of a semi-circular relation, polynomial constraint formulas can be constructed for the lines and circles that are represented in the relation L and C (i.e., given a constraint representations for the points in the finite relations L and C, (linear, respectively quadratic) constraint representations of the lines and circles they represent can be constructed). A combination of all possible sign conditions ($= 0, < 0, > 0$) on these equations gives rise to the partition of the plane induced by the lines and circles represented by the relations L and C. Using the relation P, it can be determined which parts of the partition belong to the semi-circular relation. In this way a constraint formula of a semi-circular relation can be computed from one of its LPC-representations.

So, given two LPC-representations of semi-circular relations, we can decide whether they represent the same set by first computing their constraint formulas (as described above) and next deciding whether these formulas represent the same set in $\mathbf{R}^2$. The latter can be decided using Tarski's decision procedure for the reals [35]. □

4 The language EuPL

We first define a programming language EuPL, for expressing Euclidean constructions. This language is modeled after the language of Engeler [11], with two key differences. The language of Engeler uses iteration, and as we are interested in first-order database query languages, we explore the consequences of defining a Euclidean programming language without iteration. An additional feature of EuPL is that it includes a non-deterministic choice operator. We decided to include this operator as it is used frequently in the Euclidean constructions that we want to model. However, we shall show that this choice operator is redundant, as it can be simulated by other operations under certain assumptions.

EuPL has one basic type $\langle \text{var} \rangle$ which ranges over points in the Euclidean plane. We use p, q, ... to denote variables. There is no basic notion of lines and circles: instead, lines are represented by pairs of points (p_1, p_2), and circles by triples (p_1, p_2, p_3), where (p_1, p_2) represents the line through the points p_1 and p_2 (assuming p_1 and p_2 are distinct) and (p_1, p_2, p_3) (assuming p_2 and p_3 are distinct) represents the circle with center p_1 and radius equal to $d(p_2, p_3)$, the distance between p_2 and p_3. (We could also represent circles by pairs (p_1, p_2), where p_1 is the center and p_2 a point on the circle. Our choice is arbitrary, but makes some of the examples simpler.)

Our language corresponds to one view of Euclidean constructions, as it is known that all ruler and compass constructions can be carried out on lines and circles that are represented by points (see [14] for details). The main reason for using a point representation in EuPL is to make the language similar to the database languages that are introduced in the next section, where such an encoding is necessary. We could, however, have defined a language similar to EuPL in which lines and circles are primitive notions, and all the results in the current section, with the exception of Theorem 4.5, would still hold.

The basic predicates in EuPL are:

1. **defined**(p)
2. $p_1 = p_2$
3. (a) p_1 **is on line** (p_2, p_3)
   (b) p_1 **is on circle** (p_2, p_3, p_4)
   (c) p_1 **is in circle** (p_2, p_3, p_4)
   (d) p_1 **is on the same side as** p_2 **of line** (p_3, p_4)
4. (a) **l-order** (p_1, p_2, p_3)
   (b) **c-order** (p_1, p_2, p_3, p_4)

The first condition means that the variable p represents a point. Such a test is needed, as a variable may be undefined if it is the result of an intersection of two disjoint objects (parallel lines, for example).

Under the above encoding of lines and circles, the meaning of the predicates in (3a–d) should be clear. The predicates in (4a, 4b) are order relations. The predicate **l-order** (p_1, p_2, p_3) (line-order) means that p_1, p_2 and p_3 are on the same line, and that p_2 is between p_1 and p_3. **c-order** (p_1, p_2, p_3, p_4) (circle-order) means that p_1, p_2, p_3 and p_4 are all on the same circle, in this order, either in the clockwise or in the counter-clockwise direction (see Figure 4). Note that whenever pairs (respectively triples) of points do not define lines (respectively circles), the corresponding predicates are false.

The basic operations in EuPL to compute intersections of objects correspond to the combinations line/line, line/circle and circle/circle.

1. $q :=$ **l-l-crossing** (p_1, p_2, p_3, p_4);
2. $q_1, q_2 :=$ **l-c-crossing** $(p_1, p_2, p_3, p_4, p_5)$;
3. $q_1, q_2 :=$ **c-c-crossing** $(p_1, p_2, p_3, p_4, p_5, p_6)$.

The semantics of these operators in most cases should be clear, except that the choice of which intersection point to assign to q_1 and which to q_2 is arbitrary.[2] In the case of the intersection of two parallel lines (identical or not), q is undefined. Likewise

[2]This actually introduces an additional non-determinism into the language. This can be handled by straightforward modifications of the proofs that follow, and will be ignored from now on.

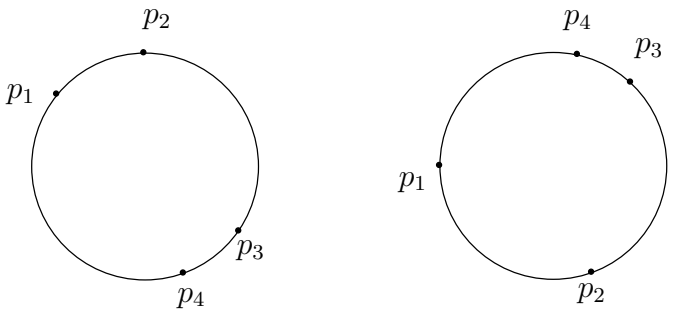


Figure 4: The two orientations for **c-order** (p_1, p_2, p_3, p_4).

for the intersection of two disjoint circles, or a disjoint circle and line. In the case of a line tangent to a circle or two circles that meet in a single point, we have $q_1 = q_2$.

The choice operator, whose syntax is

choose p **such that** $\langle$condition$\rangle$

assigns to p, in a non-deterministic manner, a point p that satisfies the given condition. When $\langle$condition$\rangle$ is unsatisfiable, p is undefined.

A formal specification of the language appears in the Appendix. The basic notion is that of a *multifunction* which has n input variables and m output variables, which represent points in the plane. Each multifunction is defined by a sequence of assignment statements and conditional statements, without looping. It's result is returned by a **result** statement.

We illustrate the language by several examples, showing how to express several Euclidean constructions in EuPL.

Example 4.1 Given a line (p_1, p_2) and a point p not on the line, construct the perpendicular (p, p') to the given line from p (see Figure 5).

multifunction $\text{perp}(p, p_1, p_2) = (p')$;
begin
choose q **such that not** (q **is on the same side as** p **of line** (p_1, p_2));
  $r, s :=$ **l-c-crossing** (p_1, p_2, p, p, q);
  $p_3, p_4 :=$ **c-c-crossing** (r, r, p, s, s, p);
  $p' :=$ **l-l-crossing** (p_1, p_2, p_3, p_4);
  result (p');
end □

The second example is introduced for the sake of later proofs. In this example, we show how to construct an arbitrary point on an ellipse. Given collinear points a, b, p and q, we construct, for each r between a and b, points r' and r'' "corresponding" to r such that (a) r' and r'' are on the ellipse through a and b with foci p and q and (b) as r ranges from a to b all the points on this ellipse are constructed. Note

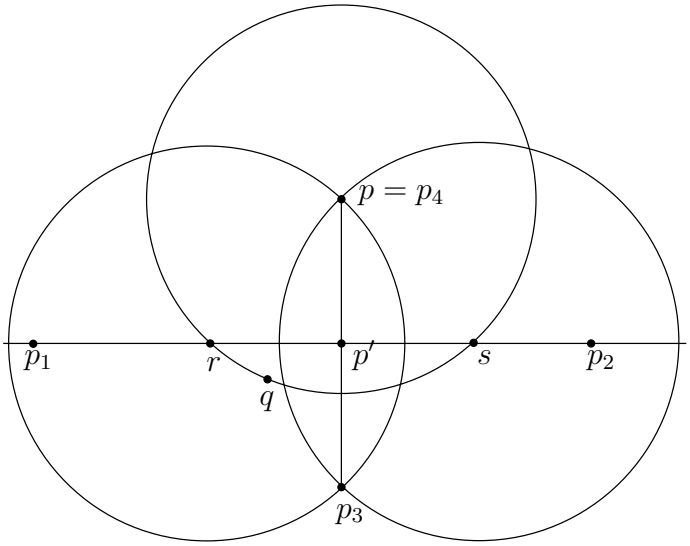


Figure 5: Construction of the perpendicular.

that the ellipse itself is not constructible with ruler and compass and therefore, by Theorem 4.2, cannot be defined by a EuPL program.

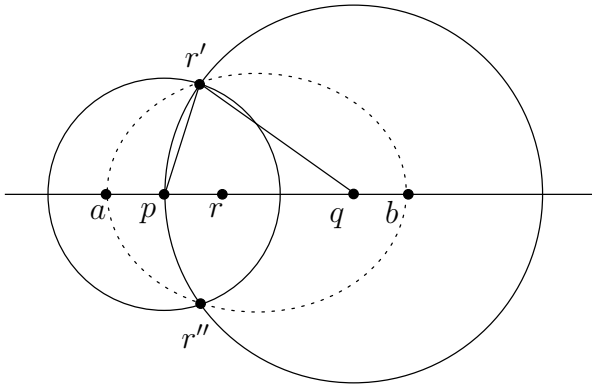


Figure 6: Construction of a point on an ellipse.

Example 4.2 Given the collinear points a, b, p, and q, with $d(a,p) = d(b,q)$, p between a and q, and q between p and b, r' is constructed as follows (see Figure 6):

multifunction put-ellipse$(a,b,p,q) = (r')$;
begin
choose r **such that** **l-order** (a,r,b);
$r', r'' :=$ **c-c-crossing** (p,a,r,q,b,r);

result (r');
end

The point r' is on the ellipse with foci p and q and major axis $d(a,b)$ since $d(p,r')+d(q,r') = d(a,r)+d(r,b) = d(p,a)+d(q,a) = d(p,b)+d(q,b)$. □

4.1 Representation Independence

As we pointed out above, EuPL is at least as powerful as a language that had lines and circles as primitives would be. But EuPL actually turns out to more powerful than desired. For example, if we are given a line represented by points (p_1,p_2), we could write a EuPL program that simply returns p_1. Such a program has no natural geometric interpretation.

The following problem therefore arises. Given a EuPL program, does the result depend on the representation of the input lines and circles or not? The inputs and outputs of a EuPL program are just points, with no indication as to what they "really" represent. We therefore have to first impose an interpretation on these points, i.e., specify which of these points represent lines or circles.

For example, given a program with inputs (p_1,p_2,p_3,p_4,p_5) and outputs (q_1,q_2), we could interpret (p_1,p_2) as a line, (p_3,p_4,p_5) as a circle, and (q_1,q_2) as a line. Other interpretations of the inputs and outputs of the program are also possible: given a specific interpretation we shall refer to P as an *interpreted* EuPL program.

The formal definition of an interpreted program is very simple. As objects (points/lines/circles) are represented by 1/2/3 points respectively, two equivalence relations are all we need.

Definition 4.1 An *interpretation* of a EuPL program P is a pair of equivalence relations E_i and E_o on the input, respectively output variables of P, such that each equivalence class has between 1 and 3 elements. □

Remark: Note that all of the language primitives are well defined, regardless of the interpretations of the variables. For example q **is in circle** (p_1,p_2,p_3) is well-defined even if (p_1,p_2) represents a line and (p_3,q) another line (though in such a case the program is unlikely to have an intuitive result, or be representation independent). We therefore do not have to address the issue of an interpretation of the internal variables of a program.

We now come to the issue of whether a EuPL program P is representation dependent or not. It turns out that, given an interpretation of P, this question is decidable.

Theorem 4.1 *It is decidable whether the result of an interpreted* EuPL *program depends on the representation of its inputs or not.*

Proof: Let P be a EuPL multifunction, with inputs $\vec{\mathbf{p}} = (p_1,\ldots,p_n)$ and outputs $\vec{\mathbf{q}} = (q_1,\ldots,q_m)$. We shall write (p^x,p^y) for the x- and y- coordinates of a point p, and shall also write expressions such as $tp = q$ and $d(p,q) = t$ for $(tp^x = q^x \wedge tp^y = q^y)$ and $(p^x)^2+(p^y)^2 = t^2$, respectively.

We define two formulas $\phi_P(\vec{\mathbf{p}}, \vec{\mathbf{q}}, \vec{\mathbf{r}})$, and $\psi_P(\vec{\mathbf{p}}, \vec{\mathbf{q}}, \vec{\mathbf{r}})$ over the theory of real closed fields, where $\vec{\mathbf{r}}$ is the list of internal variables in P. Intuitively, ϕ_P describes how $\vec{\mathbf{q}}$ depends on $\vec{\mathbf{p}}$, given $\vec{\mathbf{r}}$ as the results of the choice operations, while ψ_P describes the conditions that $\vec{\mathbf{r}}$ and $\vec{\mathbf{q}}$ must satisfy. In the following ψ_S will be taken to be a tautology, unless stated otherwise. Note that the separation into ϕ and ψ is not really needed for the current theorem, but will be used in Theorem 4.3 below. Basically, for each intersection operation we add a conjunct saying when the corresponding objects are already defined and have an intersection, and for each choice operation we add a conjunct saying that the condition is satisfiable.

The construction of ϕ and ψ is by induction using the rules defining P. We omit some of the straightforward cases.

1. When P is the sequence of statements $S_1; S_2; \ldots; S_k$, ϕ_P is defined as

   $$\phi_{S_1} \wedge \cdots \wedge \phi_{S_k}$$

   and ψ_P is

   $$\psi_{S_1} \wedge \cdots \wedge \psi_{S_k}.$$

2. Assignment statements[3]

   (a) When S is $q :=$ **l-l-crossing** (p_1, p_2, p_3, p_4), ϕ_S is

   $$(\exists! t, t')(q = tp_1 + (1-t)p_2 \wedge q = t'p_3 + (1-t')p_4)$$

   (Note that when q is undefined, this formula will be unsatisfiable).

   (b) When S is $q, q' :=$ **l-c-crossing** $(p_1, p_2, p_3, p_4, p_5)$, let $\psi_S(q, p_1, p_2, p_3, p_4, p_5)$ define q as one of the intersection points on the circle and the line, i.e.,

   $$(\exists t)(q = tp_1 + (1-t)p_2 \wedge d(q, p_3) = d(p_4, p_5))$$

   Then ϕ_S is

   $$\psi_S(q, p_1, p_2, p_3, p_4, p_5) \wedge \psi_S(q', p_1, p_2, p_3, p_4, p_5) \wedge (q \neq q' \vee (q = q' \wedge \neg(\exists q'')(\psi_S(q'', p_1, p_2, p_3, p_4, p_5) \wedge q'' \neq q)))$$

   i.e., q and q' are distinct intersection points, if such exist, and both are equal to the unique intersection point, if only one exists.

   (c) The case when S is $q, q' :=$ **c-c-crossing** $(p_1, p_2, p_3, p_4, p_5, p_6)$ is similar.

3. When S is the conditional

   if C **then** S_1 **else** S_2 **end**

   then ϕ_S is

   $$\phi_C \rightarrow \phi_{S_1} \wedge \phi_{\textbf{not } C} \rightarrow \phi_{S_2}$$

   and ψ_S is

   $$\phi_C \rightarrow \psi_{S_1} \wedge \phi_{\textbf{not } C} \rightarrow \psi_{S_2}$$

[3] As a first step in the proof, variables are renamed, such that they are not assigned more than once a value in a EuPL program.

4. When S is the choice statement

$$\textbf{choose } v \textbf{ such that } C$$

ϕ_S is a tautology whereas ψ_S is equal to ϕ_C.

5. Conditionals. Most of the conditionals, such as p_1 **is on line** (p_2, p_3) are handled in a similar way to assignments. The most complicated one is when C is **c-order** (p_1, p_2, p_3, p_4). Here ϕ_C first computes the center of the circle through p_1, p_2 and p_3, and tests whether p_4 is on this circle. If so, **c-order** (p_1, p_2, p_3, p_4) is true when p_2 and p_4 are not on the same side of the line through p_1 and p_3.

6. When C is **defined** $(\langle \text{var} \rangle)$, ϕ_C is defined as the appropriate Boolean.

Assume now that $\vec{\mathbf{p}}$ and $\vec{\mathbf{p}}'$ are two inputs to P, that are equivalent with respect to the given interpretation. Let $\vec{\mathbf{q}}$ and $\vec{\mathbf{q}}'$ be the outputs of P on these inputs. From the definition of ϕ_P and of the semantics of P, it follows that $(\exists \vec{\mathbf{r}})(\phi_P(\vec{\mathbf{p}}, \vec{\mathbf{q}}, \vec{\mathbf{r}}) \wedge \psi_P(\vec{\mathbf{p}}, \vec{\mathbf{q}}, \vec{\mathbf{r}}))$ and $(\exists \vec{\mathbf{r}}')(\phi_P(\vec{\mathbf{p}}', \vec{\mathbf{q}}', \vec{\mathbf{r}}') \wedge \psi_P(\vec{\mathbf{p}}', \vec{\mathbf{q}}', \vec{\mathbf{r}}'))$ hold.

Given the interpretation (E_i, E_o) we write formulas $\xi_i(\vec{\mathbf{p}}, \vec{\mathbf{p}}')$ and $\xi_o(\vec{\mathbf{q}}, \vec{\mathbf{q}}')$ that specify when the inputs and outputs are equivalent. For example, if $\{p_1, p_2\}$ is an equivalence class in E_i then

$$\phi_{p_1'} \textbf{ is on line } {}_{(p_1,p_2)} \wedge \phi_{p_2'} \textbf{ is on line } {}_{(p_1,p_2)} \wedge p_1' = p_2'$$

is a conjunct in ξ_i, whereas if $\{q_1, q_2, q_3\}$ is in E_o, then

$$q_1' = q_1 \wedge d(q_2', q_3') = d(q_2, q_3)$$

is a conjunct in ξ_o.

The result of P is then independent of the representation if and only if the following formula is valid

$$(\exists \vec{\mathbf{r}})((\xi_i(\vec{\mathbf{p}}, \vec{\mathbf{p}}') \wedge \phi_P(\vec{\mathbf{p}}, \vec{\mathbf{q}}, \vec{\mathbf{r}}) \wedge \psi_P(\vec{\mathbf{p}}, \vec{\mathbf{q}}, \vec{\mathbf{r}}) \wedge \phi_P(\vec{\mathbf{p}}', \vec{\mathbf{q}}', \vec{\mathbf{r}}) \wedge \psi_P(\vec{\mathbf{p}}', \vec{\mathbf{q}}', \vec{\mathbf{r}}')) \rightarrow \xi_o(\vec{\mathbf{q}}, \vec{\mathbf{q}}')).$$

As this is a first-order formula over the theory of real closed fields, the result follows from Tarski's theorem. □

Note that there were two different ways we could have interpreted the choice operator in the above theorem. Either, for equivalent inputs, we make the same choices, obtaining equivalent outputs, or we make different choices for both inputs, resulting in equivalent outputs. We have chosen the first approach above, but modifying the proof to use the latter approach is trivial. As we shall see later, "good" programs should be choice-independent anyway, so the distinction does not really matter.

4.2 Euclidean Constructions

We now compare the expressiveness of EuPL with the Euclidean constructions it is intended to model. Our first result in this direction follows directly from the definitions.

Theorem 4.2 *All* EuPL *multifunctions are constructible with ruler and compass.* □

What about the converse? The converse does not hold because our language models *first-order* ruler-and-compass constructions. For an example of a non-first-order ruler-and-compass construction, consider the following.

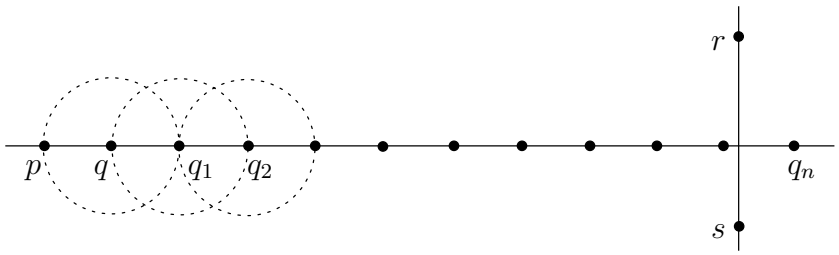


Figure 7: Non first-order construction.

Example 4.3 Let p, q, r and s be 4 different points as in Figure 7. Consider the following construction: first construct the point q_1 on the line through p and q such that $d(q_1, q) = d(q, p)$ such that $q_1 \neq p$. Then repeat this construction until we get to the other side of the line through r and s. The result will be the first point to the right of the vertical line (q_n with $n = 10$ in Figure 7). □

The computation of this point requires iteration, i.e.,

Lemma 4.1 *The above construction cannot be expressed by a* EuPL *program.*

Proof: Interpreting $d(p, q)$ as unity, we could use this construction to test whether the distance from p to the vertical line is an integer. Expressing the result of each operation as a first-order formula, this would mean that integers are definable in the theory of real-closed fields, which is known [33] to be impossible. □

The fact that we can only express first-order Euclidean constructions in EuPL may seem to detract from our original motivation, i.e., to model Euclidean geometry. In fact, some of the constructions in [24] are non first-order: for example, the construction of the gcd of two magnitudes. However, these constructions are of a numerical, rather than a geometrical, nature, and we can make the following informal claim:

Claim: All the planar *geometrical* constructions in [24] can be simulated by EuPL programs.

This claim is rather informal: by a case by case analysis, one can show that the planar geometrical construction in Euclid are all first-order.

Historical remark: For reasons of completeness we give the following overview. Specifically (ignoring the propositions that are theorems rather than constructions) this means that Propositions I/1, 2, 3, 9, 10, 11, 12, 22, 23, 31, 42, 44, 45, 46; II/11, 14; III/1, 17, 30, 33, 34;[4] IV/1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16 and VI/9, 10, 11, 12, 13, 18, 25, 28, 29, 30; X/27, 28, 29, 30, 31, 32, 33, 34, 35, 48, 49, 50, 51, 52, 53, 85, 86, 87, 88, 89, 90 are all expressible in EuPL. In contrast, some of the constructions that are essentially number-theoretic in nature (mostly deriving from Euclid's gcd algorithm), i.e., Propositions VII/2, 3, 33, 34, 36, 39; VIII/2, 4; IX/18,19; X/3, 4, are non first-order, and hence not expressible in EuPL. Propositions XI/11, 12, 23, 26, 27; XII/17; XIII/13, 14, 15, 16, 17, 18 deal with 3-dimensional constructions and are therefore outside the scope of our discussion. □

4.3 The Choice Operator

The result of the program in Example 4.1 (computation of a perpendicular) does not depend on the choice made by the choice operator. This is true for the classic Euclidean constructions as well as for all other "reasonable" EuPL programs. As with representation-independence, the question whether the result of a program depends on the results of the choice operators is decidable.

Theorem 4.3 *It is decidable whether the result of a* EuPL *program depends on the choices made by* **choose** *operators.*

Proof: Given P, we define $\phi_P(\vec{\mathbf{p}}, \vec{\mathbf{q}}, \vec{\mathbf{r}})$ and $\psi_P(\vec{\mathbf{p}}, \vec{\mathbf{q}}, \vec{\mathbf{r}})$ as in Theorem 4.1. Using these formulas, an FO + poly sentence can be written that expresses that the result of P is independent of the choices made by the **choose** operators in P. This sentence expresses that for all inputs and outputs (quantification over inputs and outputs is possible in FO + poly) and any two choices of $\vec{\mathbf{r}}$ (that satisfy some conditions) the program P satisfies the same input-output relation. Once again, the result then follows from Tarski's theorem [35]. □

Combining the proofs of Theorems 4.1 and 4.3, we obtain:

Theorem 4.4 *It is decidable whether the result of a interpreted* EuPL *program depends on the representation of its inputs and on the results of the choice operations.* □

We now show that given two points and a representation- and choice-independent program P, the use of choice is actually redundant. This means that P can be converted (effectively) into an equivalent deterministic program.

[4] In addition, Prop. 25 talks about constructing a circle from a segment of a circle. EuPL has no direct notion of a segment. Segments can be defined by a line/circle and two endpoints. This suffices for most purposes, but as a result, Prop. 25 becomes completely trivial.

Theorem 4.5 *Assuming that we have two distinct fixed points p_0 and p'_0, then every representation- and choice-independent* EuPL *program P is equivalent to a program P' which does not use the* **choose** *operator.*

Proof: Let P be an EuPL program that is representation- and choice-independent. The choice-independence of P implies that the outcome of the program does not depend on the particular value of p that is chosen in any expression

$$\textbf{choose } p \textbf{ such that } \psi(p, p_1, \ldots, p_n)$$

appearing in P. Therefore, any expression **choose** p **such that** $\psi(p, p_1, \ldots, p_n)$ appearing in P may be replaced by a series of EuPL statements, among which there is no choice statement, that produce a point p that satisfies the condition $\psi(p, p_1, \ldots, p_n)$. We shall now show how to replace the statement **choose** p **such that** $\psi(p, p_1, \ldots, p_n)$ by a series of EuPL statements that produce a point p satisfying the condition $\psi(p, p_1, \ldots, p_n)$ in a deterministic fashion.

We remark that $\psi(p, p_1, \ldots, p_n)$ is a Boolean combination of basic choice predicates in EuPL. First, we show that there exists a formula $\psi'(p, p_1, \ldots, p_n)$, equivalent to $\psi(p, p_1, \ldots, p_n)$, that is a Boolean combination of basic predicates in which the variable p appears as first variable. It then follows that p belongs to an equivalence class of the plane determined by the lines and circles (in function of the parameters $p_1, \ldots, p_n$) that appear in the formula $\psi'(p, p_1, \ldots, p_n)$. Indeed, if p appears as first variable in basic predicates, then these predicates describe how p is located with respect to certain lines and circles determined by the points $p_1, \ldots, p_n$. These lines and circles divide the plane into equivalence classes (much the same as the carrier of a semi-circular set partitions the plane – see Section 3).

We show how the variable p can be moved to the first position for one case, other cases are similar or even easier. The most difficult case is the expression p_1 **is on the same side as** p_2 **of line** (p, p_3). For points p, p_1, p_2, and p_3 that form a quadrangle, this expression is equivalent to $\neg((\varphi_1 \wedge \varphi_2) \vee (\neg\varphi_1 \wedge \neg\varphi_2))$, for with

φ_1 the expression p **is on the same side as** p_1 **of line** (p_2, p_3) and
φ_2 the expression p **is on the same side as** p_2 **of line** (p_1, p_3),

Next, we describe a EuPL program that produces a set $S_{\psi'}$ that contains at least one representative point in each of the equivalence classes of the plane determined by the lines and circles that appear in the formula $\psi'(p, p_1, \ldots, p_n)$.

The desired program, to replace **choose** p **such that** $\psi(p, p_1, \ldots, p_n)$, consists then of the computation of these representative points, checking for each of them the condition $\psi'(p, p_1, \ldots, p_n)$, and returning the first computed representative point that satisfies the condition.

To start with, $S_{\psi'}$ is the set $\{p_1, \ldots, p_n\}$. For each **c-order** (p, p_i, p_j, p_k) appearing in ψ' the center of the circle through p_i,p_j and p_k is constructed and added to $S_{\psi'}$. For each circle appearing in ψ' such that no point in $S_{\psi'}$ occurs in the circle, take the intersection of the circle and the line that connects the center to p_0 or p'_0 (p_0 and p'_0 are two fixed points that are assumed to be given), and added to $S_{\psi'}$.

Next all intersection points of all circles and lines are constructed and added to $S_{\psi'}$. This takes care of the representative points of the equivalence classes that are points. Next, for all pair of points in $S_{\psi'}$ their midpoints are added to $S_{\psi'}$; for all pairs of points on a circle the midpoints on the arc segments between them are added to $S_{\psi'}$; and for each unbounded line segment the intersection point of the segment and a circle with midpoint the point in $S_{\psi'}$ where the unbounded segment starts and radius the distance between p_0 and p_0' is added to $S_{\psi'}$. This takes care of the representative points of the equivalence classes that are 1-dimensional. Finally, for all triples of points in $S_{\psi'}$ their centroids are added to $S_{\psi'}$ This takes care of the representative points of the equivalence classes that are 2-dimensional and completes the proof. □

Based on this result, *we shall now assume that all our languages always have two distinct fixed points.* All the languages that we shall define in subsequent sections will be deterministic. We should point out that the discussion of choice operators in this section is designed to *motivate* the subsequent sections, not to apply directly to them. We have illustrated here why a language without choice operators is appropriate as a language for modeling Euclidean constructions. This will still be the case for the database languages below, even though some of our current results (such as decidability) no longer hold in the presence of a database.

To conclude, we can also use the ϕ_P and ψ_P formulas to show:

Theorem 4.6 *Equivalence is a decidable property of* EuPL *programs.*

Proof: Given the EuPL programs P_1 and P_2, we define the formulas $\phi_{P_1}(\vec{\mathbf{p}}, \vec{\mathbf{q}}, \vec{\mathbf{r}})$, $\psi_{P_1}(\vec{\mathbf{p}}, \vec{\mathbf{q}}, \vec{\mathbf{r}})$, $\phi_{P_2}(\vec{\mathbf{p}}, \vec{\mathbf{q}}, \vec{\mathbf{r}})$ and $\psi_{P_2}(\vec{\mathbf{p}}, \vec{\mathbf{q}}, \vec{\mathbf{r}})$ as in Theorem 4.1. Using these formulas, an FO + poly sentence can be written that expresses that for any input the result of P_1 on that input equals the result produced by P_2 on that input (independent of the choices made by the **choose** operators in P_1 and P_2). Since quantification over inputs and outputs (and choices) is possible in FO + poly this result then follows from Tarski's theorem [35]. □

5 The language EuQL

We now wish to define a database query language for Euclidean geometry. In this section we describe the initial attempt, EuQL, at defining such a language. EuQL is a database language and is declarative, so assignment statements are replaced by predicates. For example the crossing-point operators become predicates rather than assignments. In addition, the **defined** predicate is no longer needed, as existential quantifiers on the crossing-point predicates can be used instead.

The relations of the input database are finite 2-dimensional point relations, i.e., finite tuples of 2-dimensional points which can be represented by real polynomial formulae. The relation R_i, of arity m_i, is an m_i-ary finite 2-dimensional point relation, i.e., a $2m_i$-ary relation over the reals which can be defined by a real polynomial formula. A EuQL query over a schema $R_1, \ldots, R_n$ is of the form

$$Q(R_1, \ldots, R_n) = \{(v_1, \ldots, v_m) \mid \varphi(R_1, \ldots, R_n, v_1, \ldots, v_m)\},$$

where φ is a formula in the first-order language with equality, database predicates, all constant points with real algebraic coordinates, and the following predicates:

1. $\langle \text{var} \rangle$ **is on line** $(\langle \text{var} \rangle, \langle \text{var} \rangle)$
2. $\langle \text{var} \rangle$ **is on circle** $(\langle \text{var} \rangle, \langle \text{var} \rangle, \langle \text{var} \rangle)$
3. $\langle \text{var} \rangle$ **is in circle** $(\langle \text{var} \rangle, \langle \text{var} \rangle, \langle \text{var} \rangle)$
4. $\langle \text{var} \rangle$ **is on the same side as** $\langle \text{var} \rangle$ **of line** $(\langle \text{var} \rangle, \langle \text{var} \rangle)$
5. **l-order** $(\langle \text{var} \rangle, \langle \text{var} \rangle, \langle \text{var} \rangle)$
6. **c-order** $(\langle \text{var} \rangle, \langle \text{var} \rangle, \langle \text{var} \rangle, \langle \text{var} \rangle)$
7. $\langle \text{var} \rangle$ **is l-l-crossing point of** $(\langle \text{var} \rangle, \langle \text{var} \rangle, \langle \text{var} \rangle, \langle \text{var} \rangle)$
8. $\langle \text{var} \rangle$ **is l-c-crossing point of** $(\langle \text{var} \rangle, \langle \text{var} \rangle, \langle \text{var} \rangle, \langle \text{var} \rangle, \langle \text{var} \rangle)$
9. $\langle \text{var} \rangle$ **is c-c-crossing point of** $(\langle \text{var} \rangle, \langle \text{var} \rangle, \langle \text{var} \rangle, \langle \text{var} \rangle, \langle \text{var} \rangle, \langle \text{var} \rangle)$

The semantics of EuQL are defined as a function

$$S(Q) : \mathcal{R}_1 \times \cdots \times \mathcal{R}_n \to \mathcal{R},$$

where $\mathcal{R}_i$ is the type of relation R_i and $\mathcal{R}$ the type of the result relation of Q. The interpretations of variables, logical connectives, etc., are standard. The other predicates are interpreted as follows:

1. $S(v_1 \textbf{ is on line } (v_2, v_3))(r_1, \ldots, r_n)$ is the set of those tuples $(a_{v_1}, a_{v_2}, a_{v_3})$ for which a_{v_2} and a_{v_3} are distinct, and a_{v_1}, a_{v_2} and a_{v_3} are collinear.
2. $S(v_1 \textbf{ is on circle } (v_2, v_3, v_4))(r_1, \ldots, r_n)$ is the set of those tuples $(a_{v_1}, a_{v_2}, a_{v_3}, a_{v_4})$ for which a_{v_1} is on the circle with center a_{v_2} and radius $d(a_{v_3}, a_{v_4})$ and a_{v_3} and a_{v_4} are distinct.
3. $S(v_1 \textbf{ is in circle } (v_2, v_3, v_4))(r_1, \ldots, r_n)$ is the set of those tuples $(a_{v_1}, a_{v_2}, a_{v_3}, a_{v_4})$ for which a_{v_1} is in the interior of the circle with center a_{v_2} and radius $d(a_{v_3}, a_{v_4})$ and a_{v_3} and a_{v_4} are distinct.
4. $S(v_1 \textbf{ is on the same side as } v_2 \textbf{ of line } (v_3, v_4))(r_1, \ldots, r_n)$ is the set of those tuples $(a_{v_1}, a_{v_2}, a_{v_3}, a_{v_4})$ for which a_{v_1} is on the same side as a_{v_2} of the line (a_{v_3}, a_{v_4}), where a_{v_3} and a_{v_4} are distinct, and a_{v_2}, a_{v_3} and a_{v_4} are not collinear.
5. $S(\textbf{l-order}(v_1, v_2, v_3))(r_1, \ldots, r_n)$ is the set of those tuples $(a_{v_1}, a_{v_2}, a_{v_3})$ for which a_{v_1}, a_{v_2}, and a_{v_3} are distinct and collinear, and a_{v_1} lies between a_{v_2} and a_{v_3}.
6. $S(\textbf{c-order}(v_1, v_2, v_3, v_4))(r_1, \ldots, r_n)$ is the set of those tuples $(a_{v_1}, a_{v_2}, a_{v_3}, a_{v_4})$ for which a_{v_1}, a_{v_2}, a_{v_3} and a_{v_4} are distinct and lie on the same circle, in this order (either clockwise or counter-clockwise).

7. $S(v_1$ **is l-l-crossing point of** $(v_2, v_3, v_4, v_5))(r_1, \ldots, r_n)$ is the set of those tuples $(a_{v_1}, a_{v_2}, a_{v_3}, a_{v_4}, a_{v_5})$ for which a_{v_2} and a_{v_3} are distinct, a_{v_4} and a_{v_5} are distinct, and the lines (a_{v_2}, a_{v_3}) and (a_{v_4}, a_{v_5}) are distinct, non-parallel, and intersect at a_{v_1}.

8. $S(v_1$ **is l-c-crossing point of** $(v_2, v_3, v_4, v_5, v_6))(r_1, \ldots, r_n)$ is the set of those tuples $(a_{v_1}, a_{v_2}, a_{v_3}, a_{v_4}, a_{v_5}, a_{v_6})$ for which a_{v_2} and a_{v_3} are distinct, a_{v_5} and a_{v_6} are distinct, and a_{v_1} is a crossing point of the line (a_{v_2}, a_{v_3}) and the circle $(a_{v_4}, a_{v_5}, a_{v_6})$.

9. $S(v_1$ **is c-c-crossing point of** $(v_2, v_3, v_4, v_5, v_6, v_7))(r_1, \ldots, r_n)$ is the set of those tuples $(a_{v_1}, a_{v_2}, a_{v_3}, a_{v_4}, a_{v_5}, a_{v_6}, a_{v_7})$ for which a_{v_3} and a_{v_4} are distinct, a_{v_6} and a_{v_7} are distinct, and a_{v_1} is a crossing point of the two circles $(a_{v_2}, a_{v_3}, a_{v_4})$ and $(a_{v_5}, a_{v_6}, a_{v_7})$.

Example 5.1 Given a binary relation R that consists of pairs of points, return the unary relation with the midpoints of each tuple of R.

$$
\begin{aligned}
\{(p) \mid (\exists p_1)(\exists p_2)((&R(p_1,p_2) \wedge p_1 = p_2 \wedge p = p_1) \vee \\
&(R(p_1,p_2) \wedge \neg(p_1 = p_2) \wedge p \ \textbf{ is on line } \ (p_1,p_2) \wedge \\
&(\exists p_3)(\exists p_4)((p_3 \ \textbf{ is on circle } \ (p_1,p_1,p_2) \wedge \\
&\qquad (p_3 \ \textbf{ is on circle } \ (p_2,p_2,p_1)) \wedge \\
&\qquad (p_4 \ \textbf{ is on circle } \ (p_1,p_1,p_2)) \wedge \\
&\qquad (p_4 \ \textbf{ is on circle } \ (p_2,p_2,p_1)) \wedge \\
&\qquad \neg(p_3 = p_4) \wedge p \ \textbf{ is on line } \ (p_3,p_4)))
\end{aligned}
$$

$)\}$

□

Unfortunately, it turns out that EuQL is too powerful. To illustrate this, we show how to construct an ellipse in EuQL, and thus show that EuQL expresses more than just the Euclidean constructions. The construction is similar to the construction of an arbitrary point on an ellipse in EuPL, but by using first-order quantifiers we can essentially simulate choice operators and iterate over all possible choices.

Example 5.2 Given a 4-ary relation of points, for each tuple t, return the ellipse with foci t_1 and t_2, and major axis equal to $d(t_3, t_4)$.

$$
\begin{aligned}
\{(p) \mid (\exists t_1)(\exists t_2)(\exists t_3)(\exists t_4)(\exists q) \\
&((R(t_1,t_2,t_3,t_4) \wedge t_2 \ \textbf{ is on circle } \ (t_4,t_1,t_3) \wedge \ \textbf{l-order } \ (t_3,t_1,t_2) \wedge \\
&\ \textbf{l-order } \ (t_1,t_2,t_4) \wedge \neg(t_3 = t_4) \wedge \ \textbf{l-order } \ (t_3,q,t_4) \wedge \\
&p \ \textbf{ is on circle } \ (t_1,t_3,q) \wedge p \ \textbf{ is on circle } \ (t_2,t_4,q) \\
&))\}
\end{aligned}
$$

□

This shows that:

Theorem 5.1 EuQL *expresses queries that are not constructible in Euclidean geometry.* □

While this shows that EuQL does not match the intuition we had in mind, one might still hope that it would still serve as a language between FO + lin and FO + poly, even we if we had no intuitive feel for what could be expressed in EuQL. However, this is not the case:

Theorem 5.2 *On finite point databases,* EuQL *has the same expressive power as* FO + poly.

Proof: For the first direction, we have to show that FO + poly is at least as powerful as EuQL. Hereto it suffices to show that the predicates 1–9 above can be expressed as FO + poly-predicates. This is straightforward (and can be accomplished using techniques similar to those used in the previous section).

For the converse, it suffices to show how a polynomial constraint expression $p(x_1, \ldots, x_n)\ \theta\ 0$ can be simulated by a EuQL query. Hereto we simulate real variables by means of 2-dimensional points variables (as is made clear below). So, more precisely, we show that there exist an EuQL expression $\varphi(p_1, \ldots, p_n)$ such that $\{(p_1, \ldots, p_n) \mid \varphi(p_1, \ldots, p_n)\} = \{((x_1, 0), \ldots, (x_n, 0)) \mid p(x_1, \ldots, x_n)\ \theta\ 0\}$. If we can do this, the result will follow immediately from the fact that *(i)* it is straightforward in EuQL to compute the points $(p_1, 0)$ and $(0, p_2)$ where p_1 and p_2 are the coordinates of the point p, and vice versa, and *(ii)* both languages are first-order languages.

The proof here basically uses the same techniques that were used by Euclid to define arithmetic by geometric operations. Thereto, assume in the following that p and q are two points with coordinates of the form $(p_1, 0)$ and $(q_1, 0)$, and let o and e_1 be the origin and the the unit point on the first axis in the real plane (so, on the line connecting p and q).

First, we notice that the expression r **is l-c-crossing point of** $(o, e_1, o, o, p) \wedge \neg(r = p)$ computes the point r with coordinates $(-p_1, 0)$ and is within EuQL.

Secondly, we show how the point r with coordinates $(p_1 + q_1, 0)$ can be computed in EuQL. Without loss of generality, we may assume that $p_1 > 0$. Consider the expression x **is l-c-crossing point of** (o, e_1, p, o, q). There are two solutions for x. Which intersection point is the correct solution depends on the sign of q_1. For instance, if $q_1 > 0$, which can be decided with q **is on the same side as** (e_1, o, e_2), we choose x such that the negation of x **is on the same side as** (o, p, e_2) is satisfied. Clearly, the complete expression is in EuQL.

Next, we compute the point r with coordinates $(p_1 \cdot q_1, 0)$. While multiplication is defined by Euclid in a non-first-order way, there are well-known alternative constructions that are first-order. Let x be the point with coordinates $(p_1, q_1 - e_2)$ and let y be the point with coordinates $(0, q_1)$. These points can be computed in EuQL. Then, the point r is found as r **is l-l-crossing point of** (e_1, p, y, x).

In a similar way, we can compute the point r with coordinates $(p_1/q_1, 0)$. Let x be the point with coordinates $(q_1, 1 - p_1)$. Then $r = p_1/q_1$ can be computed from r **is l-l-crossing point of** (e_1, q, e_2, x).

We thus obtain, for any given polynomial constraint, an equivalent EuQL query, and therefore these first-order languages are equivalent. □

As Example 5.2 shows, the language EuQL is not closed on finite databases. In fact, starting from a finite input, a complete ellipse can be returned. In order to obtain the language we are looking for, we need to restrict EuQL in an appropriate way.

6 The language SafeEuQL

The purpose of this section is to isolate a set of queries which, on finite databases, are constructible in Euclidean geometry. To reach this goal, we study a subset of the EuQL queries, the SafeEuQL queries. This subset consists of the EuQL queries in safe-range normal form which satisfy a syntactically defined *safety* condition. The intuition behind the safety condition is to restrict the domain over which variables range to be finite as soon as the input database is finite. We start with the introduction of the safe-range normal form on EuQL queries. Then, we explain our imposed safety condition on EuQL queries in safe-range normal form. Finally, we show that the SafeEuQL queries can only express constructible queries.

Define a ⟨disjunction⟩ as a disjunction of ⟨conjunction⟩'s. A ⟨conjunction⟩ is defined as a conjunction of ⟨factor⟩'s. A ⟨factor⟩ is defined as a ⟨term⟩ or as $\neg$ ⟨term⟩. Finally, a ⟨term⟩ is defined as $\exists$ ⟨var⟩ (⟨disjunction⟩) or a EuQL-primitive. We call a EuQL-expression in safe-range normal form if it is defined as a ⟨disjunction⟩.

We now define the set of variables which are safe in a EuQL expression in safe-range normal form. Let R be a relation with attributes of type point. Denote the set of safe variables of an expression φ, with φ in safe-range normal form by $\mathcal{S}v(\varphi)$. The set $\mathcal{S}v(\varphi)$ then is defined as follows:

- $\mathcal{S}v(R(v_1, \ldots, v_p))$ equals $\{v_1, \ldots, v_p\}$;
- for any of the EuQL primitives φ, $\mathcal{S}v(\varphi)$ equals the empty set;
- $\mathcal{S}v((\exists v)\varphi)$ equals $\mathcal{S}v(\varphi) - \{v\}$;
- $\mathcal{S}v(\neg\varphi)$ equals the empty set;
- $\mathcal{S}v(\varphi_1 \wedge \varphi_2)$ equals the smallest set S such that the following closure properties hold:
  - if φ_i is the expression "$v_1 = v_2$" with v_1 or v_2 in S, then both v_1 and v_2 are in S (possibly v_1 and v_2 are one of the constant points);
  - if φ_i is the expression "v_1 **is l-l-crossing point of** (v_2, v_3, v_4, v_5)" and the variables $v_2, \ldots, v_5$ are in S, then v_1 is in S;
  - if φ_i is the expression "v_1 **is l-c-crossing point of** $(v_2, v_3, v_4, v_5, v_6)$" and the variables $v_2, \ldots, v_6$ are in S, then v_1 is in S;
  - if φ_i is the expression "v_1 **is c-c-crossing point of** $(v_2, v_3, v_4, v_5, v_6, v_7)$" and the variables $v_2, \ldots, v_7$ are in S, then v_1 is in S; and

– $\mathcal{S}v(\varphi_1) \cup \mathcal{S}v(\varphi_2)$ is a subset of S;

- $\mathcal{S}v(\varphi_1 \vee \varphi_2)$ equals $\mathcal{S}v(\varphi_1) \cap \mathcal{S}v(\varphi_2)$.

We are now ready to define which EuQL queries are safe.

Definition 6.1 A EuQL query $\{(v_1, \ldots, v_m) \mid \varphi(R_1, \ldots, R_n, v_1, \ldots, v_m)\}$, with φ is in safe-range normal form, is called *safe* if

(i) for each subformula of φ of the form $(\exists v)\psi$, $v \in \mathcal{S}v(\psi)$ holds, and

(ii) every free variable v_i of φ is in $\mathcal{S}v(\varphi)$. □

Example 6.1 Consider again the query which computes the midpoints of all tuples of a binary relation R. This query can be expressed with a safe EuQL query as follows:

$$\begin{aligned}\{(p) \mid\ & (\exists p_1)(\exists p_2)(p_1 = p_2 \wedge R(p_1, p_2) \wedge p = p_1) \vee \\ & (\exists p_1)(\exists p_2)(\exists p_3)(\exists p_4)(\neg(p_1 = p_2) \wedge \neg(p_3 = p_4) \wedge R(p_1, p_2) \wedge \\ & p_3 \textbf{ is c-c-crossing point of } (p_1, p_1, p_2, p_2, p_1, p_2) \wedge \\ & p_4 \textbf{ is c-c-crossing point of } (p_1, p_1, p_2, p_2, p_1, p_2) \wedge \\ & p \textbf{ is l-l-crossing point of } (p_1, p_2, p_3, p_4))\}.\end{aligned}$$

The variables p_1 and p_2 are safe in both parts of the disjunction because of the EuQL term $R(p_1, p_2)$. The variables p_3 and p_4 in the second part of the disjunction are safe since they are the two intersection points of circles defined in terms of the safe variables p_1 and p_2. Finally, p is safe because it denotes the intersection point of two lines defined by safe variables.

To illustrate that safety of a EuQL query is a purely syntactical requirement, consider the query that computes the midpoint of two points as given in Example 5.1. This time, the formula is not safe because p_3, p_4 and p are not safe. □

The set of all EuQL queries in safe-range normal form which are safe defines a query language which we shall call SafeEuQL. We have the following closure property for SafeEuQL:

Theorem 6.1 *A* SafeEuQL *query applied to a finite point database yields a finite point database which can be constructed by ruler and compass from the input.*

Proof: Let B be a finite point database and φ a SafeEuQL expression. We show that, when φ is applied on B, every variable v with $v \in \mathcal{S}v(\varphi)$ ranges over a finite domain.

First, let φ be quantifier-free. We proof the above claim on the safe variable v in φ by induction on the length of φ, i.e., on the number of propositional connectives in φ.

For the basis of this induction, we observe that the only SafeEuQL expressions which can have v as a safe variable are of the form $R(\ldots, v, \ldots)$, $v = c_1$,

v **is l-l-crossing point of** (c_1, c_2, c_3, c_4), v **is l-c-crossing point of** $(c_1, c_2, c_3, c_4, c_5)$, or v **is c-c-crossing point of** $(c_1, c_2, c_3, c_4, c_5, c_6)$ with $c_1, \ldots, c_6$ safe. By assumption the relation R is finite, and thus the claim becomes trivial.

Now assume that the claim holds for safe variables in quantifier-free SafeEuQL expressions of length at most k. Let v be a safe variable in the quantifier-free SafeEuQL expression φ of length $k+1$. We distinguish two cases:

1. $\varphi \equiv \psi_1 \vee \psi_2$. From the definition of safety it follows that v is safe in both ψ_1 and ψ_2. By the induction hypotheses, we know that for ψ_1 and ψ_2 applied on B, v ranges over finite domains, say D_1 and D_2, respectively. Then, clearly, for φ applied on B, v ranges also over a finite domain bounded by $D_1 \cup D_2$.

2. $\varphi \equiv \psi_1 \wedge \psi_2$. Denote S the union of $\mathcal{S}v(\psi_1)$ and $\mathcal{S}v(\psi_2)$. When ψ_1 and ψ_2 are applied on B, by the induction hypotheses, every variable of S ranges over a finite domain. Let D be the union of all these finite domains of variables in S. Now repeat the following process until v is in S. Consider every SafeEuQL primitive in φ which do not occur in $\neg\psi$, with $\neg\psi$ a subformula of ψ_1 or ψ_2. If the primitive is of the form $v_1 = v_2$ with $v_1 \in S$, then add v_2 to S. If the primitive has the form v_1 **is l-l-crossing point of** (v_2, v_3, v_4, v_5), v_1 **is l-c-crossing point of** $(v_2, v_3, v_4, v_5, v_6)$, or v_1 **is c-c-crossing point of** $(v_2, v_3, v_4, v_5, v_6, v_7)$ with $\{v_2, \ldots, v_7\} \subseteq S$, then add v_1 to S and let D' be the finite set of crossing-points obtained by letting the variables $v_2, \ldots, v_7$ range over the points of D. Augment D with D'. The resulting domain D is still finite and every variable of S ranges over at most the points in D. Since, by assumption, v is safe in φ and from the definition of safety of a variable, it follows that this process terminates within a finite number of steps. Thus, for φ applied on B, v ranges over a finite set of points.

We emphasize that the case $\varphi \equiv \neg\psi$ cannot occur since this expression has no safe variables, in contradiction with the assumption that v is safe in φ.

Next, let φ contain (existential) quantifiers. Then consider a SafeEuQL-term of the form $(\exists v)\psi$ with ψ quantifier-free. By definition of safety, v has to be safe in ψ. As a consequence of the first part of the proof, when ψ is applied on B, v ranges over a finite domain, say D_v. Now replace in φ the formula $(\exists v)\psi$ by $D_v(v) \wedge \psi$, which results in a SafeEuQL expression with the same result on B as φ. Since φ has only a finite number of quantifiers, we can repeat this quantifier elimination process until we obtain a quantifier-free SafeEuQL expression with the same result as φ on the finite point database B. All (free) variables in this expression range over a finite domain, and thus the result of the expression will also be finite.

Finally, we observe that for the given points, every EuQL primitive can be simulated with ruler and compass. Since every variable in a SafeEuQL expression applied on a finite point database ranges over a finite set of points, there exists, for a finite point database, a finite sequence of ruler-and-compass constructions which yields the same set of points as the SafeEuQL expression applied on the finite point database. Thus, for every SafeEuQL expression, the finite output database can be constructed with compass and ruler from the input database, which concludes the proof. □

We conclude this section with the following result:

Theorem 6.2 *We have full arithmetical power on the coordinates of safe variables in* SafeEuQL*, i.e., we can subtract, add, multiply, and divide coordinates of safe variables.*

Proof: Assume that p and q are safe variables. It is possible in SafeEuQL to compute the points with coordinates $(p_1, 0)$, $(0, p_2)$, $(q_1, 0)$, and $(0, q_2)$ where p_1, p_2, q_1, and q_2 are the coordinates of the points p and q, respectively. So, without loss of generality, we assume that p and q are safe variables with coordinates of the form $(p_1, 0)$ and $(q_1, 0)$. It can be easily seen that all variables in the EuQL expressions for negation, addition, multiplication and division, given in the proof of Theorem 5.2, are safe, which concludes the proof. □

7 The main results

In this section, we define two query languages which are closed on semi-circular relations. The first, SafeEuQL$^{\uparrow}$, captures those first-order geometrical constructions that can be described by ruler and compass. The second captures all FO + poly expressible queries that map semi-circular relations to semi-circular relations.

As such, we lift the query language SafeEuQL, which is defined on finite point databases, to a general query language (namely SafeEuQL$^{\uparrow}$), which works on semi-circular databases. Since SafeEuQL works on finite point databases, we interpret these SafeEuQL$^{\uparrow}$ queries to work on intensional representations of semi-circular databases (for the LPC-representation of semi-circular databases we refer back to Section 3).

To achieve our goal, we show that there exists an encoding and corresponding decoding of semi-circular databases into finite point databases. The following lemmas show that this encoding and decoding are expressible in FO + poly.

We then prove various theorems that compare the expressive power of these languages with FO + lin and FO + poly.

We use the following convention: R_{poly} refers to a 2-dimensional semi-algebraic relation, R_{circ} refers to a semi-circular relation, and R_{lin} refers to a 2-dimensional semi-linear relation.

Given an LPC-database (see Section 3) which, by definition, consists of finite relations of points in the plane, there exists a database consisting of three relations containing the coordinates of the points in the relations L, P, and C respectively. Indeed, for every point appearing in the relations L, P, or C, we can compute the coordinates of that point with respect to the coordinate system defined by the three constant points o, e_1, and e_2 by ruler and compass. This computation adds up to the construction of parallel lines with the line oe_2 (respectively oe_1) through the points in the finite relations, and then taking the intersection of these lines with the line oe_1 (respectively oe_2). Remark that points the relations L, P, and C have real algebraic coordinates. Conversely, given two real algebraic coordinates, we can compute the

point in the plane corresponding to these coordinates. Also remember that EuQL contains constants for all points in the plane with real algebraic coordinates.

In the following, we will not distinguish between the point and coordinate representation of an LPC-database, i.e., given L, P, and C relations, we will interpret them point or coordinate-wise depending on the context in which they are used. For example, if we say that a semi-circular database can be mapped to an LPC-database, we mean that the semi-circular database can be mapped to the coordinate representation of an LPC-database.

7.1 The query language SafeEuQL$^{\uparrow}$

Before defining SafeEuQL$^{\uparrow}$, we need two lemmas.

Lemma 7.1 *There exists an* FO + poly *query* $Q_{(L,P,C)\to R_{\text{circ}}}$ *that maps the coordinate representation of every intensional LPC-representation of a semi-circular relation to the semi-circular relation it represents.*

Proof: Assume that the predicate $LSign(p_x, p_y, q_x, q_y, x_1, y_1, x_2, y_2)$ evaluates to true when the points with coordinates (x_1, y_1) and (x_2, y_2) belong to the same class of the partition induced by the line defined by (p_x, p_y) and (q_x, q_y), i.e., when the points with coordinates (x_1, y_1) and (x_2, y_2) are either on the line, or on the same side of the line. In the same way, denote by $CSign(p_x, p_y, q_x, q_y, x_1, y_1, x_2, y_2)$ the predicate which evaluates to true if the points with coordinates (x_1, y_1) and (x_2, y_2) belong to the same class of the partition induced by the circle with midpoint (p_x, p_y) and through the point with coordinates (q_x, q_y), i.e., when the points with coordinates (x_1, y_1) and (x_2, y_2) are both either inside the circle, or on the circle, or outside the circle. Both predicates *LSign* and *CSign* can be easily expressed within FO + poly. The query $Q_{(L,P,C)\to R_{\text{circ}}}$ now is expressible in the language FO + poly as follows. Let (p_x, p_y) be the coordinates of a point of the P relation. The expression $(\forall q_x)(\forall q_y)(\forall r_x)(\forall r_y)(L(q_x, q_y, r_x, r_y) \to LSign(q_x, q_y, r_x, r_y, p_x, p_y, x, y) \wedge C(q_x, q_y, r_x, r_y) \to CSign(q_x, q_y, r_x, r_y, p_x, p_y, x, y))$ evaluates to true if the points with coordinates (p_x, p_y) and (x, y) belong to the same class of the partition of the plane induced by the lines and circles defined in the L and C relations. Clearly, the union of all partition classes of points in the P relation defines the corresponding semi-circular set. □

Lemma 7.2 *There exists an* FO + poly *query*

$$Q_{R_{\text{circ}}\to(L,P,C)}$$

that maps any semi-circular relation to the coordinate representation of an intensional LPC-representation of this relation.

Proof: First, we show how the relations $\tilde{L}$ and $\tilde{C}$ containing the line and circle carriers of the semi-circular set S, can be computed within FO + poly. We observe that the L and C relations containing representation points of the carriers in $\tilde{L}$ and

$\tilde{C}$ respectively, can be easily obtained from $\tilde{L}$ and $\tilde{C}$ as follows. For every line in $\tilde{L}$, we obtain representation points as the intersection points with the coordinate-axis. For lines parallel with the x-axis, representation points are found as intersection points with the lines defined by $x = 0$ and $x = 1$. Similarly, for lines parallel with the y-axis, the representation points consist of the intersection points with the lines defined by $y = 0$ and $y = 1$. For every circle in $\tilde{C}$, the center and the leftmost point of the circle are considered as representation points of the circle.

Consider now the computation of $\tilde{L}$ and $\tilde{C}$. Now, consider the topological boundary of S, ∂S, and the topological closure of S minus S, $\bar{S} - S$. These one dimensional semi-circular figures contain all information necessary to compute the carriers of S [9].

Therefore, for every line-segment of ∂S and $\bar{S} - S$ the supporting line is computed and added to the relation $\tilde{L}$. Similarly, for every circle segment detected in ∂S and $\bar{S} - S$, the supporting circle is added to the relation $\tilde{C}$. The corresponding $\mathsf{FO} + \mathsf{poly}$ expressions are omitted.

Next, all isolated points of ∂S and $\bar{S} - S$ are computed by $\mathsf{FO} + \mathsf{poly}$expressions. Then, endpoints of half-lines, line segments, and circle segments of ∂S and $\bar{S} - S$ are computed as follows. Let F be a one dimensional semi-circular set. The point p is an endpoint of a line segment or half-line of F, if it is the beginning point of a line segment which is completely contained in F and this line segment can not be extended such that it contains the point p and still is completely contained in F. Similarly, the point p is an endpoint of a circle segment of F, if it is the beginning point of a circle segment which is contained in F and this circle segment can not be extended such that is contains p and still is completely contained in F. Through every isolated point and endpoint of a half-line, line segment, or circle segment, lines parallel with the coordinate-axis are added to $\tilde{L}$. Now, $\tilde{L}$ contains all line carriers of the semi-circular set S.

So, $\tilde{L}$ and $\tilde{C}$ contain all carriers of the semi-circular set S.

Now, we turn to the computation of the P relation, the set of representatives of each class of the partition induced by $\tilde{L}$ and $\tilde{C}$ which is contained in S. For every point of the plane, the partition class with respect to $\tilde{L}$ and $\tilde{C}$ is computed. Then, for each partition class, a representative is found and added to the P relation.

As shown in Lemma 7.1, for every point of the plane the partition class can be computed in $\mathsf{FO} + \mathsf{poly}$. We now compute all values of d such that the open cube with side d and centered around the origin, intersects all classes of the partition induced by L and C. The result will be a half-line of the form $a < d$ with a a constant. Denote B the open cube with side $a + 1$ and centered around the origin. Then B has a non-empty intersection with all partition classes. Let p be a point of the plane and denote by X the intersection of the partition class of p and B. Every representative of the bounded set X is also a representative of the partition class of p. We distinguish three cases based on the dimension of X. In the zero dimensional case, X is a point and can be added to the relation P without further computing. If X is one dimensional, then X can consist of a complete circle, an open circle segment, or a finite number of open line segments. Since X is bounded, half-lines can not occur. For every line and circle segment of X, the midpoint can be computed

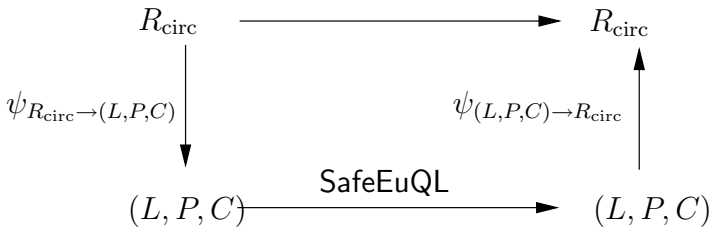


Figure 8: The query language $\mathsf{SafeEuQL}^{\uparrow}$ is closed on semi-circular relations.

and added to P as representative of X. For a complete circle, we add the leftmost point of the circle to P. Finally, we investigate the two dimensional case. We obtain a representative of the bounded region denoted by X as follows. First, project X on the x-axis. The result is a one dimensional bounded open set X_1. Let r_x be the x-coordinate of the midpoint of the line segment of X_1 closest to the origin. Next, compute the intersection of the line $y = r_x$ and X. Finally, project this intersection on the y-axis, and let r_y be the y-coordinate of the midpoint of the line segment closest to the origin. Add the point with coordinates (r_x, r_y) to P as representative of X, and thus of the partition class of p. This completes the proof. □

Definition 7.1 We say that a query Q, that maps semi-circular relations to semi-circular relations, belongs to the language $\mathsf{SafeEuQL}^{\uparrow}$ if it is a composition of the form

$$Q_{(L,P,C)\to R_{\text{circ}}} \circ Q_{\mathsf{SafeEuQL}} \circ Q_{R_{\text{circ}}\to(L,P,C)},$$

with Q_{SafeEuQL} a $\mathsf{SafeEuQL}$ query[5] (see Figure 8). More precisely, the $\mathsf{SafeEuQL}^{\uparrow}$ queries are a composition of three queries. First, the query which maps a semi-circular relation to its LPC representation. The point representation of this LPC-database then is the input of a $\mathsf{SafeEuQL}$ query which produces an LPC-database as a result. Finally, the coordinate representation of this LPC-database is mapped to the semi-circular relation it represents. □

$Q_{(L,P,C)\to R_{\text{circ}}}$ and $Q_{R_{\text{circ}}\to(L,P,C)}$ are syntactically well-defined, and so are the $\mathsf{SafeEuQL}$ queries. It would be desirable, however, to find a more elegant syntactic definition of $\mathsf{SafeEuQL}^{\uparrow}$.

The language $\mathsf{SafeEuQL}^{\uparrow}$ has the closure property on the class of semi-circular relations. This is illustrated in Figure 8.

We thus have a syntactically defined subclass of $\mathsf{FO} + \mathsf{poly}$ that is closed on semi-circular relations.

[5]To be more precise, Q_{SafeEuQL} is the product of three $\mathsf{SafeEuQL}$ queries.

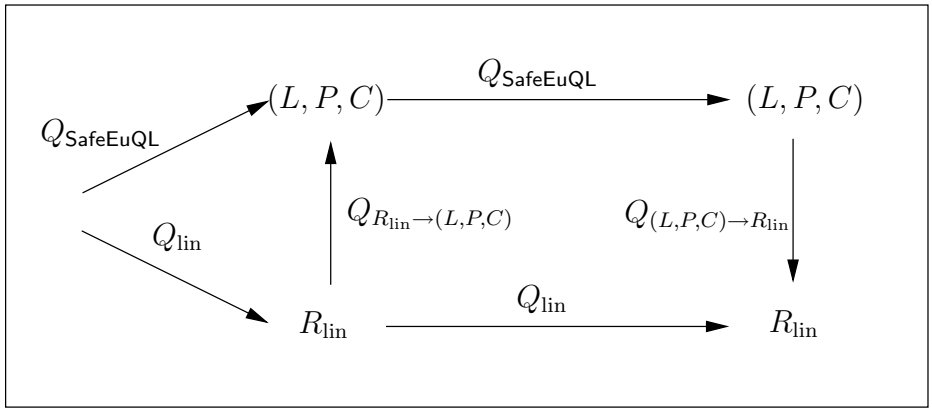


Figure 9: Any $\mathsf{FO}+\mathsf{lin}$ query on semi-linear relations can be simulated in SafeEuQL on the intensional level. The two arrows at the left denote the property that any semi-linear relation can be defined in the language $\mathsf{FO}+\mathsf{lin}$ and that any (L,P,C) database can be defined in SafeEuQL.

7.2 On semi-linear relations the language SafeEuQL$^{\uparrow}$ is more expressive than $\mathsf{FO}+\mathsf{lin}$

As discussed in Section 3 and Lemma 7.2, every two-dimensional semi-linear relation can be intensionally represented as a finite LPC-database. We will show that every $\mathsf{FO}+\mathsf{lin}$ query on semi-linear relations can be simulated in SafeEuQL on the intensional level. Therefore, we can conclude that SafeEuQL$^{\uparrow}$, on semi-linear databases, can express every $\mathsf{FO}+\mathsf{lin}$ query. On the other hand, it is clear that SafeEuQL$^{\uparrow}$ is more expressive than $\mathsf{FO}+\mathsf{lin}$ on linear inputs, simply because the latter can only output linear databases and the former also semi-circular ones.

This is shown in more detail in Figure 9 and stated more precisely in

Theorem 7.1 *There exist* $\mathsf{FO}+\mathsf{poly}$ *queries* $Q_{R_{\mathrm{lin}}\to(L,P,C)} : R_{\mathrm{lin}} \mapsto (L,P,C)$ *and* $Q_{(L,P,C)\to R_{\mathrm{lin}}} : (L,P,C) \mapsto R_{\mathrm{lin}}$ *such that, for every* $\mathsf{FO}+\mathsf{lin}$ *query* $Q_{\mathrm{lin}} : R_{\mathrm{lin}} \mapsto R_{\mathrm{lin}}$, *there exists a* SafeEuQL *query* $Q_{\mathsf{SafeEuQL}} : (L,P,C) \mapsto (L,P,C)$ *such that*

$$Q_{\mathrm{lin}} = Q_{(L,P,C)\to R_{\mathrm{lin}}} \circ Q_{\mathsf{SafeEuQL}} \circ Q_{R_{\mathrm{lin}}\to(L,P,C)}.$$

The proof of this theorem is far from evident. The main challenge here is to prove the existence of the SafeEuQL query which simulates a given $\mathsf{FO}+\mathsf{lin}$ query. From Lemma 7.1 and Lemma 7.2, it follows that there exists $\mathsf{FO}+\mathsf{poly}$ queries which translate a linear relation into the coordinate representation of a corresponding LPC-database, and vice versa. Moreover, an examination of the proofs of Lemma 7.1 and Lemma 7.2 shows that the corresponding LPC-database for a linear relation has an empty C-relation, as one might intuitively expect. We will therefore refer to it as an LP-database in the following.

The major difficulty in simulating an FO + lin expression by a SafeEuQL expression is that, in general, subformulas of FO + lin expressions operate in higher dimensional space since quantifiers are allowed in linear formulas. Therefore, the LP-representation technique for two dimensional linear relations has to be generalized (in the obvious way) to allow the representation of higher dimensional linear relations. For any semi-linear set of $\mathbf{R}^n$, there exist a finite number of n-dimensional hyperplanes which partition $\mathbf{R}^n$ into topological open, convex cells, such that a finite number of these cells constitute the given semi-linear set. The $(n-1)$-dimensional hyperplanes are finitely represented with n linear independent points. An n-dimensional point p can be represented within SafeEuQL as a tuple of n two dimensional points as follows $((p_1, 0), \ldots, (p_n, 0))$ with p_i the ith coordinate of p. Based on this, each n-dimensional semi-linear set will be represented in SafeEuQL with a pair of relations (H^n, P^n) with H^n a $2n^2$-ary relation containing the representation of a finite number of $(n-1)$-dimensional hyperplanes, and with P a $2n$-ary relation containing the representation of the representatives of the partition classes constituting the semi-linear set.

Before we concentrate on the proof of Theorem 7.1, we prove the following two lemmas:

Lemma 7.3 *Denote by H^n the $2n^2$-ary point relation containing the representation of a finite number of hyperplanes of the n-dimensional space. Assume that x and y are safe variables. Then, there exists a* SafeEuQL *expression $SameSide(H^n; p, q)$ which decides whether the two n-dimensional points p and q are on the same side of each hyperplane of H^n, i.e., the points p and q belong to the same partition class induced by the hyperplanes of H^n.*

Proof: Given n linear independent points defining a hyperplane of $\mathbf{R}^n$, we can compute the Cartesian equation of the form $a_1x_1 + \cdots + a_nx_n = 0$ of this hyperplane: each coefficient a_i can be obtained from the coordinates of the points defining the hyperplane, which are safe points, with only multiplication, division, subtraction, and addition. To decide whether two safe points p and q are on the same side of this hyperplane, we have to compare the sign of the real values $a_1p_1 + \cdots + a_np_n$ and $a_1q_1 + \cdots + a_nq_n$ using **is on the same side as** . From Theorem 6.2 is follows that the predicate $SameSide(H; p, q)$ is indeed expressible in SafeEuQL. □

Lemma 7.4 *Denote H^n the $2n^2$-ary point relation containing the representation of a finite number of hyperplanes of the n-dimensional space. There exists a* SafeEuQL *expression which computes the relation P^n that contains for every partition class induced by the hyperplanes of H^n at least one representation point.*

Proof: First, add the representation of every coordinate-plane of the n-dimensional space to the relation H^n. The partition induced by the hyperplanes of this new relation H^n is a refinement of the partition induced by the old relation H^n, and, thus, a finite set of representatives of this new partition is also a set of representatives of

the old partition. Starting from the unit points o, e_1, and e_2, we can clearly compute in SafeEuQL the finite representation of each coordinate-plane.

Next, we take n hyperplanes from the relation H^n which are linear independent. Two hyperplanes are linear independent if they are not parallel or collapse. A set of hyperplanes is linear independent if each pair of hyperplanes is linear independent. We can decide in SafeEuQL whether two hyperplanes are linear independent as follows. Let p be a representation point of the first hyperplane and q a representation point of the second hyperplane. Now, if for each representation point r of the first hyperplane, the point obtained as $r + q - p$ belongs to the second hyperplane, we can conclude that the two hyperplanes are parallel or collapse. Since p, q and r have to be in H^n, i.e., are safe variables, we can use Theorem 6.2 to obtain a SafeEuQL expression which computes $r + q - p$. Lemma 7.3 guarantees that it can be decided in SafeEuQL whether a point belongs to a given hyperplane. So, all ingredients necessary to decide whether a set of hyperplanes is linear independent are SafeEuQL computable.

Each set of n linear independent hyperplanes of the n dimensional space intersect in exactly one point. We don't work out the details here, but, since we have full arithmetical power on the coordinates of the points defining the hyperplanes (see Theorem 6.2), it should be clear that there exists a SafeEuQL expression which computes the intersection point of these n linear independent hyperplanes.

Denote I the set of intersection points of all sets of n linear independent hyperplanes of H^n. Remark that this set can not be empty since each hyperplane of the original relation H^n intersects at least $n-1$ coordinate planes, and, thus, contributes at least one point to I.

The finite set of points I now contains enough material to compute a representation point of each *bounded* partition class induced by the hyperplanes of H^n. Each bounded partition class is convex, since it is the intersection of a finite number of open half-planes. Therefore, the topological closure of a partition class can be written as the convex hull of a finite number of points, the so called corner points. All these corner points, however, are contained in I. A representative can then be found as the barycenter of these corner points. Thus, the set of points which contains the barycenter of each tuple of n points (duplicate points are allowed) of I, has a representative for each bounded partition class induced by the hyperplanes of H^n. The computation of the barycenter of n safe points is SafeEuQL expressible by Theorem 6.2.

For unbounded partition classes, the set I does not necessarily contain enough material to compute a representative of these classes. To solve this problem, we introduce the notion of a "bounding box". The rationale behind this bounding box is that each partition class induced by the hyperplanes of H^n has a non-empty intersection with this bounding box, and, thus, to compute a representative of a partition class, it suffices to compute a representative of the intersection of that partition class with the bounding box which is, of course, bounded.

We now continue with the computation of this bounding box.

For each coordinate plane of the n-dimensional space, we compute two hyperplanes parallel with this coordinate plane and such that all points of I are in between

these hyperplanes. Again, the finite representation of these two hyperplanes can be easily computed in SafeEuQL. Assume, for instance, we want to compute the hyperplane perpendicular on the i-axis such that all points of I are to the "left" of this hyperplane. Let p be a point of I. Add one (subtract one when searching for the hyperplane where all points of I are to the "right" of this hyperplane) to the i-coordinate of p and call the new point p'. Translate the coordinate-plane perpendicular to the i-axis to obtain the hyperplane through p' and perpendicular to the i-axis. If all points of I are on the same side of this hyperplane, we have found the hyperplane we were seeking. All necessary arithmetic can be done on points belonging to I and H^n, and, as such, is expressible in SafeEuQL.

Denote by H_B the set of $2n$ hyperplanes which are obtained as illustrated above. Denote by B the topological open n dimensional bounding box defined by the hyperplanes of H_B. We claim that the open box B has a non-empty intersection with each partition class induced by the hyperplanes of H^n. Indeed, each unbounded partition class has at least one corner point: it intersects at least $n-1$ coordinate planes which were added to H^n. This corner point is obtained from intersection of hyperplanes of H^n, and, thus, this corner point is contained in B. Since B is topological open, there exists a neighborhood of the corner point which is completely contained within B. The corner point, however, also belongs to the topological closure of its partition class. As such, this neighborhood of the corner point has a non-empty intersection with the partition class. So, B has a non-empty intersection with the partition class. Finally, bounded partition classes are completely contained within B, since their topological closure can be written as the convex hull of points of I.

Let I' be the set of all intersection points of n linear independent hyperplanes of $H^n \cup H_B$. It has been shown earlier in this proof that the computation of I' can be done within SafeEuQL. The finite set of points P^n containing the barycenter of each n tuple of points from I', contains for each partition class induced by the hyperplanes of H^n a representative of the intersection of that partition class with B. Since each partition class has a non-empty intersection with B, the set P^n contains a representative for each partition class induced by the hyperplanes of H^n. □

With these lemmas, we are able to prove Theorem 7.1, namely that there exist $\mathsf{FO}+\mathsf{poly}$ queries $Q_{R_{\text{lin}}\to(L,P,C)} : R_{\text{lin}} \mapsto (L,P,C)$ and $Q_{(L,P,C)\to R_{\text{lin}}} : (L,P,C) \mapsto R_{\text{lin}}$ such that, for every $\mathsf{FO}+\mathsf{lin}$ query $Q_{\text{lin}} : R_{\text{lin}} \mapsto R_{\text{lin}}$, there exists a SafeEuQL query $Q_{\mathsf{SafeEuQL}} : (L,P,C) \mapsto (L,P,C)$ such that

$$Q_{\text{lin}} = Q_{(L,P,C)\to R_{\text{lin}}} \circ Q_{\mathsf{SafeEuQL}} \circ Q_{R_{\text{lin}}\to(L,P,C)}.$$

Proof of Theorem 7.1: Assume Q_{lin} is an $\mathsf{FO}+\mathsf{lin}$ query defined by a formula φ of $\mathsf{FO}+\mathsf{lin}$. Let I_{lin} be an arbitrary two-dimensional linear input relation to Q_{lin} and let O_{lin} be the result of Q_{lin} applied on I_{lin}.

Denote by I_L and I_P, and O_L and O_P the LP-representations for I_{lin} and O_{lin}, respectively, which can be computed in $\mathsf{FO}+\mathsf{poly}$ as explained in Lemma 7.2. In the following we will show that there exists SafeEuQL queries Q_L and Q_P such that for any input relation I_{lin} with LP-representation I_L and I_P, $Q_L(I_L,I_P) = O_L$ and $Q_P(I_L,I_P) = O_P$ such that O_L and O_P are an LP-representation of the output $Q_{lin}(I_{lin}) = O_{lin}$.

We show the existence of these SafeEuQL queries Q_L and Q_P that simulate the FO + lin query expressed by the formula φ by induction on the structure of φ.

This means we have to show the following.

Induction proof: This proof has a part for atomic formulas and a part for composed formulas.

- For atomic FO + lin formulas, i.e., for formulas of the form $I_{lin}(x, y)$ and $\sum_{i=1}^{n} a_i x_i \ \theta \ 0$, with $\theta \in \{=, <, >\}$ there exist SafeEuQL queries Q_L and Q_P that simulate them;
- For FO + lin formulas of the form $\varphi_1(x_1, \ldots, x_m) \vee \varphi_2(x_1, \ldots, x_n)$, $\neg\varphi_1(x_1, \ldots, x_m)$ and $(\exists x_i)\varphi_1(x_1, \ldots, x_m)$ there exist SafeEuQL queries Q_L and Q_P that simulate them, assuming the existence of such SafeEuQL formulas has already been shown for the formulas $\varphi_1(x_1, \ldots, x_m)$ and $\varphi_2(x_1, \ldots, x_n)$. □

We now start the induction proof.

• *Atomic formula of the form $I_{lin}(x, y)$:* For formulas of the form $I_{lin}(x, y)$, we introduce SafeEuQL expressions which compute the relations H_I^2 and P_I^2. For each tuple (p, q) of I_L, H_I^2 contains a tuple of the form $((p_x, 0), (p_y, 0), (q_x, 0), (q_y, 0))$ where p_x, p_y, q_x, q_y are the coordinates of the two dimensional points p and q. For each tuple (p) of I_P, the relation P_I^2 contains a tuple $((p_x, 0), (p_y, 0))$ with p_x and p_y the coordinates of the two dimensional point p. Clearly, these computations can be done within SafeEuQL.

• *Atomic formula of the form $\sum_{i=1}^{n} a_i x_i \ \theta \ 0$, with $\theta \in \{=, <, >\}$:* For formulas of the form $\sum_{i=1}^{n} a_i x_i \ \theta \ 0$, we remark that there exist n linear independent points $p_1, \ldots, p_n$ (in the n-dimensional space) such that the smallest affine space containing $p_1, \ldots, p_n$ is precisely the hyperplane given by $\sum_{i=1}^{n} a_i x_i = 0$. Furthermore, there also exists a point p which satisfies the formula $\sum_{i=1}^{n} a_i x_i \ \theta \ 0$. All these points $p, p_1, \ldots, p_n$ can be found using algebraic computation techniques and have real algebraic coordinates since the a_i's are real algebraic too. Therefore, there exists SafeEuQL formulas which compute the coordinates of $p, p_1, \ldots, p_n$. The relations H^n and P^n denoting the LP-representation of the semi-linear set described by $\sum_{i=1}^{n} a_i x_i \ \theta \ 0$ can now be easily computed using SafeEuQL.

This finishes the induction proof for atomic FO + lin formulas. We now proceed with composed formulas.

So, assume that the claim holds for the FO + lin formulas $\varphi_1(x_1, \ldots, x_m)$ and $\varphi_2(x_1, \ldots, x_n)$. By the induction hypotheses, we may assume that there exists SafeEuQL formulas which compute the representations (H_1^m, P_1^m) and (H_2^n, P_2^n) of φ_1 and φ_2, respectively. We then have to show that we can compute in SafeEuQL the representations of the formulas $\varphi_1(x_1, \ldots, x_m) \vee \varphi_2(x_1, \ldots, x_n)$, $\neg\varphi_1(x_1, \ldots, x_m)$, and $(\exists x_i)\varphi(x_1, \ldots, x_m)$ to complete the induction proof.

• *Composed formula of the form $\varphi_1(x_1, \ldots, x_m) \vee \varphi_2(x_1, \ldots, x_n)$:* If $m \neq n$, assume without loss of generality $m < n$. We start with showing how the representation (H^n, P^n) of the formula $\varphi_1(x_1, \ldots, x_m) \vee \varphi_2(x_1, \ldots, x_n)$ can be computed. Therefore, in this case, we need the representation (H_1^n, P_1^n) of the formula

$\varphi_1(x_1, \ldots, x_m, \ldots, x_n)$ in the n-dimensional space. This representation, however, can be easily computed from (H_1^m, P_1^m). Let $(p_0, \ldots, p_m)$ be a tuple of m-dimensional points of H_1^m. We extend every point p_i with $n-m$ zero coordinates (in the obvious way), and we take the $n-m$ extra points defined as $(p_0, 1, 0, \ldots, 0), (p_0, 0, 1, 0 \ldots, 0)$, $\ldots, (p_0, 0, \ldots, 0, 1)$ to obtain a tuple of n-dimensional points of the relation H_1^n. The set of representatives P_1^n is obtained from P_1^m by adding $n-m$ zero coordinates to every m-dimensional point p of P_1^m.

We now turn to the computation of the representation (H^n, P^n) of the formula $\varphi_1(x_1, \ldots, x_n) \vee \varphi_2(x_1, \ldots, x_n)$. The relation H^n is easily found as the union of H_1^n and H_2^n. Denote P the set of representatives of all partition cells induced by the hyperplanes represented by H^n. The set P^n is then obtained from P with the following SafeEuQL formula: $\{x \mid P(x) \wedge (\exists y)(P_1^n(y) \wedge \mathit{SameSide}(H_1^m; x, y)) \vee (P_2^n(y) \wedge \mathit{SameSide}(H_2^m; x, y))\}$.

• *Composed formula of the form* $\neg\varphi_1(x_1, \ldots, x_m)$*:* The computation of the representation (H^m, P^m) of the formula $\neg\varphi_1(x_1, \ldots, x_m)$ is rather easy. Let P be the set of representatives of all partition cells of the partition induced by the hyperplanes represented by H_1^m, then $H^m = H_1^m$ and $P^m = \{x \mid P(x) \wedge \neg(\exists y)(P_1^m(y) \wedge \mathit{SameSide}(H_1^m; x, y))\}$.

• *Composed formula of the form* $(\exists x_i)\varphi_1(x_1, \ldots, x_m)$*:* Finally, we have to compute the representation (H^{m-1}, P^{m-1}) of the formula $(\exists x_i)\varphi_1(x_1, \ldots, x_m)$.

Let H^m and P^m be the representation for φ_1. For each two hyperplanes of H^m, we compute the finite representation of the intersection of these hyperplanes. Then, we compute the projection of this finite representation. If the projection of two points coincides, we introduce a arbitrary new point, such that in the end we have $(m-1)$ linear independent points which denote a hyperplane in the ith-coordinate plane. Add these $(m-1)$ linear independent points as a tuple to H^{m-1}. This concludes the computation of H^{m-1}. Denote P the set of all representatives of the partition induced by the hyperplanes in H^{m-1}. Let p be a point of P and let q be a point of P^m. Compute the intersection point r of the line through the point p and perpendicular to the ith-coordinate-plane, and the hyperplane through q and parallel with the ith-coordinate plane. If q en r belong to the same partition class induced by the hyperplanes of H^m, then add p to P^{m-1}. This concludes the computation of P^{m-1}. All techniques necessary to do the above computations are already shown to be in SafeEuQL.

This completes the induction proof.

The two SafeEuQL queries obtained as such compute a relation of the form $\{((p_1, 0), (p_2, 0), (q_1, 0), (q_2, 0)) \mid O_L((p_1, p_2), (q_1, q_2))\}$ on the one hand, and a relation of the form $\{((p_1, 0), (p_2, 0)) \mid O_P((p_1, p_2))\}$ on the other hand. From these relations, the relations O_L and O_P can be easily computed in SafeEuQL. □

7.3 On both semi-circular and semi-linear relations, the language FO + poly is more expressive than SafeEuQL$^{\uparrow}$

In this section, we first define the fragment of FO + poly that maps semi-circular relations to semi-circular relations. Later on, we will show that this language also allows

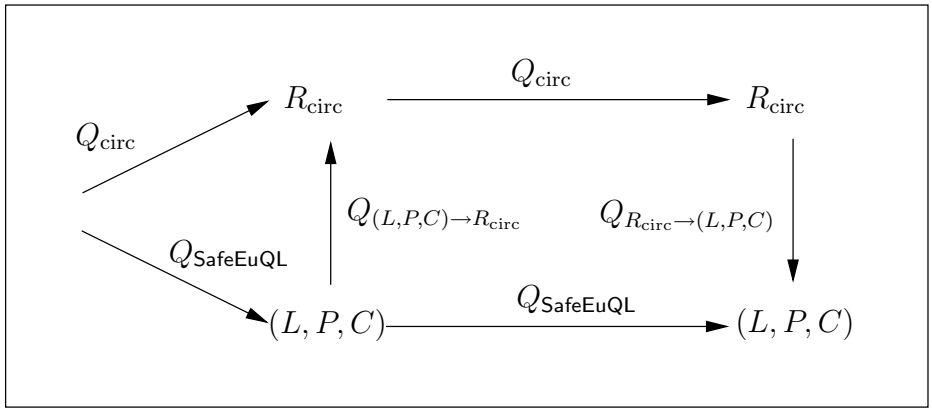


Figure 10: The query languages $\mathsf{SafeEuQL}^{\uparrow}$ and $\mathsf{FO}+\mathsf{poly}_{\text{circ}}$. Again, the arrows at the left denote which relations and databases can be defined in the respective languages.

for the formulation of "non-constructible" queries and therefore is more powerful than $\mathsf{SafeEuQL}^{\uparrow}$.

Definition 7.2 We denote the set of $\mathsf{FO}+\mathsf{poly}$ queries that map semi-circular relations to semi-circular relations by $\mathsf{FO}+\mathsf{poly}_{\text{circ}}$. □

The following theorem expresses that $\mathsf{SafeEuQL}^{\uparrow}$ is a strict subset of $\mathsf{FO}+\mathsf{poly}_{\text{circ}}$.

Theorem 7.2 (Figure 10) *For every* SafeEuQL *query* $Q_{\mathsf{SafeEuQL}} : (L,P,C) \mapsto (L,P,C)$*, there exists an* $\mathsf{FO}+\mathsf{poly}_{\text{circ}}$ *query* $Q_{\text{circ}} : R_{\text{circ}} \mapsto R_{\text{circ}}$ *such that*

$$Q_{\mathsf{SafeEuQL}} = Q_{R_{\text{circ}}\to(L,P,C)} \circ Q_{\text{circ}} \circ Q_{(L,P,C)\to R_{\text{circ}}},$$

but not conversely.

Proof: First, we show the existence of the $\mathsf{FO}+\mathsf{poly}_{\text{circ}}$ query Q_{circ}. ¿From Theorem 5.2, it follows that every query expressible in EuQL, can be simulated in $\mathsf{FO}+\mathsf{poly}$. Since SafeEuQL is a sub-language of EuQL, the same holds for the former. Let $\tilde{Q}_{\text{circ}}$ be the $\mathsf{FO}+\mathsf{poly}$ query which simulates the SafeEuQL query Q_{SafeEuQL}, i.e., $\tilde{Q}_{\text{circ}}$ applied to the coordinate-representation of an LPC-database has the same result as Q_{SafeEuql} applied to the LPC-database. Then let Q_{circ} be the query $Q_{(L,P,C)\to R_{\text{circ}}} \circ \tilde{Q}_{\text{circ}} \circ Q_{R_{\text{circ}}\to(L,P,C)}$. Clearly, Q_{circ} is an $\mathsf{FO}+\mathsf{poly}_{\text{circ}}$ query which satisfies the conditions as stated above.

For the second part, consider the query that maps a semi-circular relation consisting of a line segment qr and a point p (illustrated in Figure 11) that is not collinear with q and r to that relation augmented with two line segments ps and pt such that the angles $\angle pqs$, $\angle pst$, and $\angle ptr$ are equal. This query is expressible in

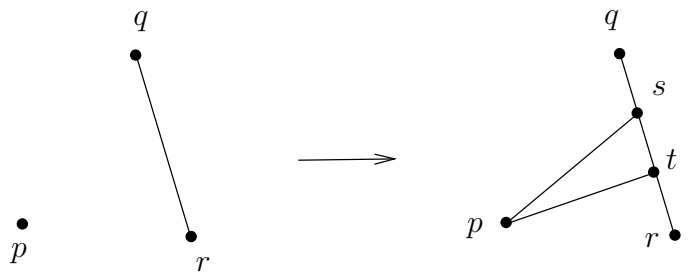


Figure 11: The trisection query.

$\mathsf{FO} + \mathsf{poly}$. Since the query maps every semi-circular relation to a semi-circular relation, it belongs to $\mathsf{FO} + \mathsf{poly}_{\mathrm{circ}}$. It is not expressible in $\mathsf{SafeEuQL}^{\uparrow}$, however, since the trisection of an angle cannot be done with ruler and compass, and is therefore not expressible in $\mathsf{SafeEuQL}$. □

We conclude with a remark on $\mathsf{FO} + \mathsf{poly}$ which is defined on R_{poly} relations. The richer class of 2-dimensional figures on which $\mathsf{FO} + \mathsf{poly}$ is defined allows us to express, for example, the construction of an ellipse. Once restricted to semi-circular relations, however, it follows immediately from the definitions that $\mathsf{FO} + \mathsf{poly}$ and $\mathsf{FO} + \mathsf{poly}_{\mathrm{circ}}$ have the same expressive power.

7.4 Conclusion

Figure 12 summarizes our results.

- On the bottom level of Figure 12, we have $\mathsf{FO} + \mathsf{lin}$ as a query language on semi-linear relations. We recall that queries concerning Euclidean distance are not expressible in this language. Not only does the data model only allow semi-linear relations, but, moreover, there are $\mathsf{FO} + \mathsf{poly}$ queries mapping semi-linear relations to semi-linear relations that are not expressible in $\mathsf{FO} + \mathsf{lin}$. The transformation of a relation into its convex hull is an example [37].

- On the next level, we have more expressive power on (the intensional representation of) semi-linear relations. We can also express queries that involve Euclidean distance. The data model also supports a class of relations wider than the semi-linear ones. All queries expressible in $\mathsf{SafeEuQL}$ are constructible by ruler and compass. So, the trisection of a given angle, for instance, is *not* expressible in $\mathsf{SafeEuQL}$.

- In $\mathsf{FO} + \mathsf{poly}_{\mathrm{circ}}$, we gain in expressive power compared to the previous level. For example, trisection of an angle is expressible in this language. The language $\mathsf{FO} + \mathsf{poly}_{\mathrm{circ}}$ has the same expressive power as $\mathsf{FO} + \mathsf{poly}$ on semi-circular relations.

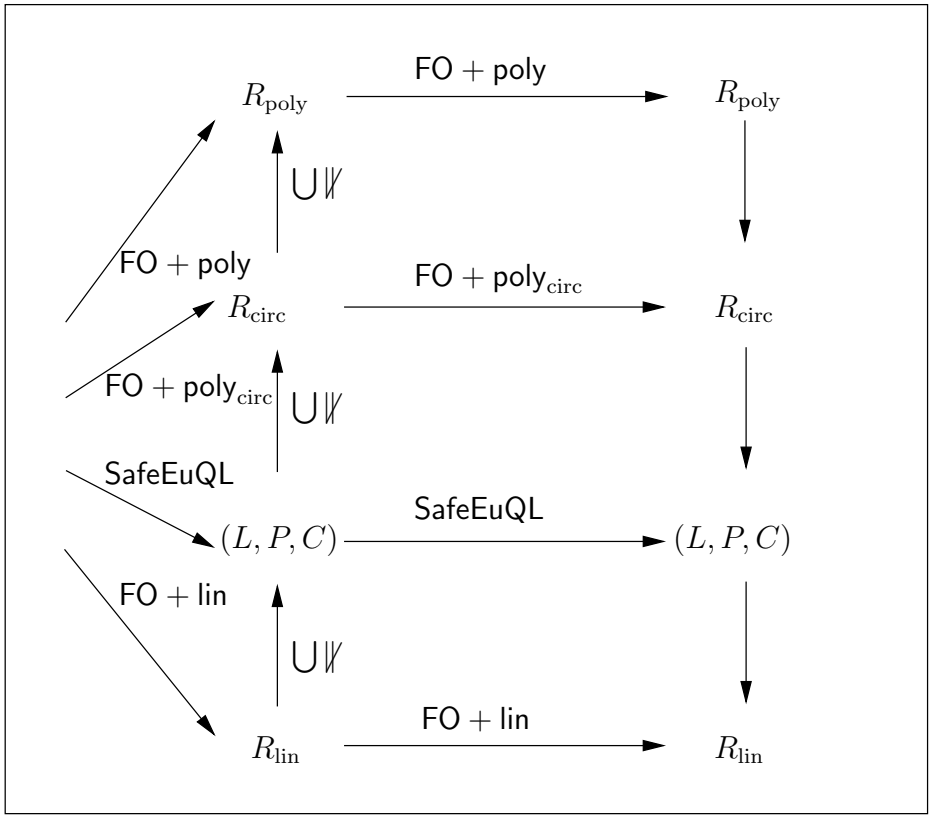


Figure 12: Comparison of the different query languages.

- On the top level, we have $\mathsf{FO} + \mathsf{poly}$. Here, the data model supports all relations definable with polynomial constraints, including queries (e.g,. construction of an ellipse) that are not expressible in $\mathsf{FO} + \mathsf{poly}_{\mathrm{circ}}$.

References

[1] D. Abel and B.C. Ooi (eds.), *Proceedings of the 3rd international symposium on spatial databases*, Lecture Notes in Computer Science, vol. 692, Berlin, Springer-Verlag, 1993.

[2] F. Afrati, T. Andronikos, and T. Kavalieros, *On the expressiveness of first-order constraint languages*, Proceedings of the 1st Workshop on Constraint Databases and their Applications (Berlin) (G. Kuper and M. Wallace, eds.), Lecture Notes in Computer Science, vol. 1034, Springer-Verlag, 1995, pp. 22–39.

[3] F. Afrati, S. Cosmadakis, S. Grumbach, and G. Kuper, *Linear versus polynomial constraints in database query languages*, Proceedings of the 2nd International Workshop on Principles and Practice of Constraint Programming (Berlin) (A. Borning, ed.), Lecture Notes in Computer Science, vol. 874, Springer-Verlag, 1994, pp. 181–192.

[4] M. Benedikt and L. Libkin, *Safe constraint queries*, Proceedings of the 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (New York), ACM Press, 1998, pp. 99–108.

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

[6] A. Buchmann (ed.), *Proceedings of the 1st international symposium on spatial databases*, Lecture Notes in Computer Science, vol. 409, Berlin, Springer-Verlag, 1989.

[7] B.F. Caviness and J.R. Johnson (eds.), *Quantifier elimination and cylindrical algebraic decomposition*, Springer-Verlag, Wien, New York, 1998.

[8] G.E. Collins, *Quantifier elimination for real closed fields by cylindrical algebraic decomposition*, Automata Theory and Formal Languages (Berlin) (H. Brakhage, ed.), Lecture Notes in Computer Science, vol. 33, Springer-Verlag, 1975, pp. 134–183.

[9] F. Dumortier, M. Gyssens, L. Vandeurzen, and D. Van Gucht, *On the decidability of semi-linearity for semi-algebraic sets and its implications for spatial databases*, Proceedings of the 16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (New York), ACM Press, 1997, pp. 68–77.

[10] M.J. Egenhofer and J.R. Herring (eds.), *Proceedings of the 4th international symposium on spatial databases*, Lecture Notes in Computer Science, vol. 951, Berlin, Springer-Verlag, 1995.

[11] E. Engeler, *Remarks on the theory of geometrical constructions*, The Syntax and Semantics of Infinitary Languages (Berlin) (A. Dold and B. Echraun, eds.), Lecture Notes in Mathemathics, vol. 72, Springer-Verlag, 1968, pp. 64–76.

[12] ———, *Foundations of mathematics*, Springer-Verlag, Berlin, 1992.

[13] Euclid, *Euclides elementorum geometricorum lib. xv*, Basileae: Apud Iohannem Hervagium, 1537.

[14] H. Eves, *College geometry*, Jones and Barlett Publishers, Boston, 1995.

[15] S. Grumbach, *Implementing linear constraint databases*, Proceedings of the 2nd Workshop on Constraint Databases and Applications (Berlin) (V. Gaede, A. Brodsky, O. Günther, D. Srivastava, V. Vianu, and M. Wallace, eds.), Lecture Notes in Computer Science, vol. 1191, Springer-Verlag, 1997, pp. 105–115.

[16] S. Grumbach, P. Rigaux, M. Scholl, and L. Segoufin, *Dedale, a spatial constraint database*, Sixth International Workshop on Database Programming Languages—DBPL'97, Lecture Notes in Computer Science, vol. 1369, Springer-Verlag, 1998, pp. 124–135.

[17] S. Grumbach and J. Su, *Finitely representable databases*, Proceedings of the 13th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (New York), ACM Press, 1994, pp. 289–300.

[18] ———, *Towards practical constraint databases*, Proceedings of the 15th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (New York), ACM Press, 1996, pp. 28–39.

[19] ———, *Queries with arithmetical constraints*, Theoretical Computer Science **173** (1997), no. 1, 151–181.

[20] S. Grumbach, J. Su, and C. Tollu, *Linear constraint query languages: Expressive power and complexity*, Proceedings of the Logic and Computational Complexity Workshop (D. Leivant, ed.), Lecture Notes in Computer Science, vol. 960, Springer-Verlag, 1994, pp. 426–446.

[21] O. Günther and H.-J. Schek (eds.), *Proceedings of the 2nd international symposium on spatial databases*, Lecture Notes in Computer Science, vol. 525, Berlin, Springer-Verlag, 1991.

[22] R. H. Güting (ed.), *Advances in spatial databases—6th symposium (ssd'99)*, Lecture Notes in Computer Science, vol. 1651, Berlin, Springer-Verlag, 1999.

[23] M. Gyssens, L. Vandeurzen, and D. Van Gucht, *An expressive language for linear spatial database queries*, Proceedings of the 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (New York), ACM Press, 1998, pp. 109–118.

[24] T. Heath, *The thirteen books of euclid's elements*, Dover, New York, 1956.

[25] D. Hilbert, *Grundlagen der geometrie*, Springer-Verlag, Berlin, 1930.

[26] J.E. Hopcroft and J.D Ullman, *Introduction to automata theory, languages, and computation*, Addison-Wesley, Reading, Massachusetts, 1979.

[27] P.C. Kanellakis, G.M. Kuper, and P.Z. Revesz, *Constraint query languages*, Journal of Computer and System Sciences **51** (1995), 26–52.

[28] J. L. Lassez, *Querying constraints*, Proceedings of the 9th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (New York), ACM Press, 1990, pp. 288–298.

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

[30] J. Paredaens, G. Kuper, and L. Libkin (eds.), *Constraint databases*, Springer-Verlag, 2000.

[31] M.F. Preparata and M.I. Shamos, *Computational geometry*, Springer-Verlag, New York, 1985.

[32] J. Renegar, *On the computational complexity and geometry of the first-order theory of the reals*, Journal of Symbolic Computation **13** (1989), 255–352.

[33] Julia Robinson, *Definability and decision problems in arithmetic.*, Journal of Symbolic Logic **14** (1949), 98–114.

[34] M. Scholl and A. Voisard (eds.), *Proceedings of the 5th international symposium on spatial databases*, Lecture Notes in Computer Science, vol. 1262, Berlin, Springer-Verlag, 1997.

[35] A. Tarski, *A decision method for elementary algebra and geometry*, University of California Press, Berkeley, 1951.

[36] L. Vandeurzen, M. Gyssens, and D. Van Gucht, *On the desirability and limitations of linear spatial query languages*, Proceedings of the 4th International Symposium on Spatial Databases (Berlin) (M.J. Egenhofer and J.R. Herring, eds.), Lecture Notes in Computer Science, vol. 951, Springer-Verlag, 1995, pp. 14–28.

[37] ———, *On query languages for linear queries definable with polynomial constraints*, Proceedings of the 2nd International Conference on Principles and Practice of Constraint Programming (Berlin) (E.C. Freuder, ed.), Lecture Notes in Computer Science, vol. 1118, Springer-Verlag, 1996, pp. 468–481.

Appendix: Formal Specification of EuPL.

The formal specification of EuPL is as follows. The basic notion is that of a "multifunction", a function that takes a fixed number of input points, and constructs a fixed (possibly more than one) number of output points.

⟨multifunction⟩ →
multifunction ⟨name⟩ '(' ⟨var⟩ (, ⟨var⟩) ∗ ')'
= '(' ⟨type⟩ (, ⟨type⟩) ∗ ')';
begin
⟨statement⟩ (; ⟨statement⟩)∗
end

⟨choice-condition⟩ →
true | false |
⟨var⟩ = ⟨var⟩ |
⟨var⟩ **is on line** '(' ⟨var⟩ , ⟨var⟩ ')' |
⟨var⟩ **is on circle** '(' ⟨var⟩ , ⟨var⟩ , ⟨var⟩ ')' |
⟨var⟩ **is in circle** '(' ⟨var⟩ , ⟨var⟩ , ⟨var⟩ ')' |
⟨var⟩ **is on the same side as** ⟨var⟩ **of line** '(' ⟨var⟩ , ⟨var⟩ ')' |
l-order '(' ⟨var⟩ , ⟨var⟩ , ⟨var⟩ ')' |
c-order '(' ⟨var⟩ , ⟨var⟩ , ⟨var⟩ , ⟨var⟩ ')' |
⟨choice-condition⟩ **and** ⟨choice-condition⟩ |
⟨choice-condition⟩ **or** ⟨choice-condition⟩ |
not ⟨choice-condition⟩ |

Eu-conditions are those used in **if** clauses. They are slightly more general that the conditions used in choice statements.

⟨eu-condition⟩ →
⟨choice-condition⟩ |
defined (⟨var⟩) |
⟨eu-condition⟩ **and** ⟨eu-condition⟩ |
⟨eu-condition⟩ **or** ⟨eu-condition⟩ |
not ⟨eu-condition⟩ ⟨statement⟩ →
⟨empty statement⟩ |
⟨assignment⟩ |
⟨conditional statement⟩ |
⟨choice⟩ |
⟨result⟩
⟨empty statement⟩ →
⟨assignment⟩ →
⟨var⟩ := **l-l-crossing** (⟨var⟩ , ⟨var⟩ , ⟨var⟩ , ⟨var⟩) |
⟨var⟩ , ⟨var⟩ := **l-c-crossing** (⟨var⟩ , ⟨var⟩ , ⟨var⟩ , ⟨var⟩ , ⟨var⟩) |
⟨var⟩ , ⟨var⟩ := **c-c-crossing** (⟨var⟩ , ⟨var⟩ , ⟨var⟩ , ⟨var⟩ , ⟨var⟩ , ⟨var⟩) |

⟨conditional statement⟩ →
    if ⟨eu-condition⟩
    then ⟨statement⟩(;⟨statement⟩)*
    else ⟨statement⟩(;⟨statement⟩)*
    end
⟨choice⟩ →
    choose ⟨var⟩ **such that** ⟨choice-condition⟩
⟨result⟩ →
    result ⟨var⟩ (, ⟨var⟩)∗