SIAM J. COMPUT.
Vol. 36, No. 6, pp. 1570–1599

FIRST-ORDER LANGUAGES EXPRESSING CONSTRUCTIBLE SPATIAL DATABASE QUERIES[*]

BART KUIJPERS[†], GABRIEL KUPER[‡], JAN PAREDAENS[§], AND LUC VANDEURZEN[†]

Abstract. The research presented in this paper is situated in the framework of constraint databases introduced by Kanellakis, Kuper, and Revesz in their seminal paper of 1990, specifically, the language with real polynomial constraints ($\mathsf{FO} + \mathsf{poly}$). For reasons of efficiency, this model is implemented with only *linear* polynomial constraints, but this limitation to linear polynomial constraints has severe implications on the expressive power of the query language. In particular, when used for modeling spatial data, important queries 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 and allow the expression of queries concerning distance. We seek inspiration in the Euclidean constructions, i.e., constructions by ruler and compass. We first present a programming language that captures exactly the first-order ruler-and-compass constructions that are expressible in a first-order language with real polynomial constraints. If this language is extended with a **while** operator, we obtain a language that is complete for all ruler-and-compass constructions in the plane. We then transform this language in a natural way into a query language on finite point databases, but this language turns out to have the same expressive power as $\mathsf{FO} + \mathsf{poly}$ and is therefore too powerful for our purposes. We then consider a *safe* fragment of this language and use this to construct a query language that allows the expression of Euclidean distance without having the full power of $\mathsf{FO} + \mathsf{poly}$.

Key words. constraint databases, real algebraic geometry, first-order logic, query languages, ruler-and-compass constructions

AMS subject classifications. 03B70, 14P10, 51M04, 57R05, 68P15

DOI. 10.1137/S0097539702407199

1. Introduction and motivation.

Kanellakis, Kuper, and Revesz [28, 29] (see also [30]) introduced the framework of *constraint databases* which provides a rather general model for spatial databases [32]. Spatial database systems [1, 7, 10, 21, 22, 37] 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. These are called *semi-algebraic* sets [6].

Several query languages on databases using the constraint model have been stud-

[*]Received by the editors May 7, 2002; accepted for publication (in revised form) August 28, 2006; published electronically February 20, 2007.
http://www.siam.org/journals/sicomp/36-6/40719.html

[†]Theoretical Computer Science Group, Hasselt University and Transnational University of Limburg, B-3590 Diepenbeek, Belgium (bart.kuijpers@uhasselt.be, luc.vandeurzen@groept.be).
[‡]Dipartimento Informatica e Telecommunicazioni, Università di Trento, I-38050 Povo, Trento, Italy (kuper@acm.org).
[§]Department of Mathematics and Computer Science, University of Antwerp, Middelheimlaan 1, B-2020 Antwerp, Belgium (jan.paredaens@ua.ac.be).

ied. One such query language is obtained by extending the relational calculus with polynomial inequalities [32]. 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

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

Although variables in such expressions range over the real numbers, queries expressed in this calculus can still be computed effectively. In particular, the closure property holds: Any $\mathsf{FO}+\mathsf{poly}$ query, when evaluated on a spatial database in the constraint model, yields a spatial database in the constraint model. This follows immediately from Tarski's quantifier-elimination procedure for the first-order theory of real closed fields [38].

However, Tarski's quantifier-elimination procedure is computationally very expensive. Since the 1970s various more efficient algorithms have been proposed, including Cylindrical Algebraic Decomposition [8, 9], which are still unsuitable for practice. The best known algorithms were proposed in the 1990s [4, 34], and, by the use of alternative data structures [14], in the best case, quantifier elimination is exponential in the number of quantifier blocks. (A recent textbook on this matter is [36], and for a discussion on the influence of data structures we refer the reader to [25]). Due to this complexity it seems to be infeasible for real spatial database applications to rely on quantifier elimination (we discuss this further below). In existing implementations of the constraint model, such as the DEDALE system [15, 16, 17], the constraints are restricted to *linear* polynomial constraints, and 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 [17, 40]. Indeed, in one of the main application domains, geographical information systems, linear approximations are used to model geometrical objects (for an overview of this field since the early '90s, see [1, 7, 10, 21, 22, 37]).

When we extend the relational calculus with linear polynomial inequalities, we obtain an effective language which has the same closure property as above but this time with respect to linear databases. We refer to this language as $\mathsf{FO}+\mathsf{lin}$, and therefore 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 for both $\mathsf{FO}+\mathsf{poly}$ and $\mathsf{FO}+\mathsf{lin}$ the cost of quantifier elimination grows exponentially with the number of blocks of quantifiers to be eliminated, an argument in favor of the language $\mathsf{FO}+\mathsf{lin}$ is that there exists a conceptually "easier" way (Fourier's method [27, 31]) to eliminate quantifiers for this language which makes an effective implementation feasible [15, 16, 17]. This algorithm has the same asymptotic complexity as the quantifier-elimination procedure for $\mathsf{FO}+\mathsf{poly}$, though there is a slight gain in data complexity: Grumbach and Su have shown that the data complexity for $\mathsf{FO}+\mathsf{lin}$ is NC^1, while it is NC for $\mathsf{FO}+\mathsf{poly}$ (for certain restrictions of $\mathsf{FO}+\mathsf{lin}$, namely ℓ-bounded instances, an AC^0 bound is obtained) [19]. From the practical point of view, a more significant advantage of the linear model is the existence of numerous efficient algorithms for geometrical operations [33].

There are, however, a number of serious disadvantages to the restriction to linear polynomial constraints, related to 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, 18, 20, 40, 41] 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 goal of this paper is to overcome these limitations of $\mathsf{FO}+\mathsf{lin}$ for the special case of two-dimensional spatial databases.

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. Vandeurzen, Gyssens, and Van Gucht [40, 41] 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. However, even this extension does not allow the expression of queries involving distance.

In this paper, we define a new query language, $\mathsf{SafeEuQL}^{\uparrow}$, and a class of two-dimensional constraint databases on which this language is closed, called *semi-circular* databases. 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, 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 is strictly less powerful than $\mathsf{FO}+\mathsf{poly}$.

To define this language, 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 13 books of his *Elements* [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. For a 20th century description of these constructions and the main results concerning them, we refer the reader to [26].

An alternative to considering languages based on ruler and compass constructions would be to use a constraint language based on the field of the constructible real numbers. It follows from a result by Ziegler that this theory is undecidable [42], however. Ziegler showed, among other results, that any finitely axiomatizable subtheory of the reals with addition, multiplication, and order is undecidable (as conjectured by Tarski).

We define and study in the current paper three languages for ruler-and-compass constructions.

First, we define a programming language that describes Euclidean constructions. We will refer to this procedural language as EuPL (short for Euclidean Programming Language). Engeler [11, 12] studied a similar programming language in the '60s, but his language contains a while-loop and therefore goes beyond first-order logic-based languages. His language also differs from ours in that EuPL also contains a choice statement. This statement corresponds to choosing arbitrary points, which 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: that equivalence and satisfiability of EuPL programs are decidable, and that it is decidable whether a program is deterministic.

We then transform the programming language EuPL into a query language for finite point databases, called EuQL (short for Euclidean Query Language). It turns out that this calculus can express nonconstructible queries—in fact, we show that EuQL has the same expressive power on finite point databases as FO + poly. 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 databases to their representation and back. 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 [5] and Gyssens, Vandeurzen, and Van Gucht [23].

We then compare the expressive power of SafeEuQL$^\uparrow$ with the expressive power of FO + poly on semi-circular databases, and show that on semi-linear databases FO + poly is more expressive than 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. A horizontal arrow indicates that a language is closed on the given class of databases. A subscheme of the form

$$\begin{tikzcd} B \arrow[rr, "\mathcal{L}_1"] & {} & B \\ A \arrow[rr, "\mathcal{L}_2"] & {} \arrow[u, "\subsetneq"'] & A \end{tikzcd}$$

means that on databases in the class A, the language $\mathcal{L}_1$, mapping databases in class A to databases in class A, is more expressive than the language $\mathcal{L}_2$, mapping databases in class B to databases in class B (where B is a larger class of databases than A). The top part of Figure 1, for instance, expresses that SafeEuQL$^\uparrow$ is strictly more expressive than FO + poly on semi-circular databases.

Organization of the paper. In the next section, we define FO + poly and FO + lin. In section 3, we introduce the class of 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 that of FO + lin on semi-linear databases and FO + poly on semi-circular databases.

$$\begin{tikzcd}
\{\text{semi-algebraic databases}\} \arrow[rr, "\mathsf{FO}+\mathsf{poly}"] & {} & \{\text{semi-algebraic databases}\} \\
& {} \arrow[u, "\subsetneq"] & \\
\{\text{semi-circular databases}\} \arrow[rr, "\mathsf{SafeEuQL}^{\uparrow}"] & {} \arrow[u] & \{\text{semi-circular databases}\} \\
& {} \arrow[u, "\subsetneq"] & \\
\{\text{semi-linear databases}\} \arrow[rr, "\mathsf{FO}+\mathsf{lin}"] & {} \arrow[u] & \{\text{semi-linear databases}\} \\
& {} \arrow[u, "\subsetneq"] & \\
\{\text{finite point databases}\} \arrow[rr, "{\mathsf{EuQL}, \mathsf{SafeEuQL}}"] & {} \arrow[u] & \{\text{finite point databases}\}
\end{tikzcd}$$

FIG. 1. *Comparison of the expressive power of the different languages.*

2. Constraint-based database models. In this section, we provide the necessary background for the polynomial and linear constraint database models and formally define two 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 submodel 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. A *polynomial formula* is a well-formed first-order logic formula over 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

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

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 shall 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 [38], 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*, and a query in the polynomial database model is a computable mapping from m-tuples of semi-algebraic relations to a semi-algebraic relation.

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 2.1. 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 [28, 29] (this property is commonly referred to as "closure").

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, for eliminating quantifiers in the linear model (this method is described in [27, 31]).

A *linear database* is a finite set of *semi-linear relations*. 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 [27, 31, 30].

Example 2.2. 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 2.1, 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, and then describe an encoding of these relations as finite relations of points. This encoding will be *complete*, meaning that every such relation has an encoding, and *lossless*, meaning that the original relation can be recovered from the encoding (even in $\mathsf{FO} + \mathsf{poly}$).

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

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

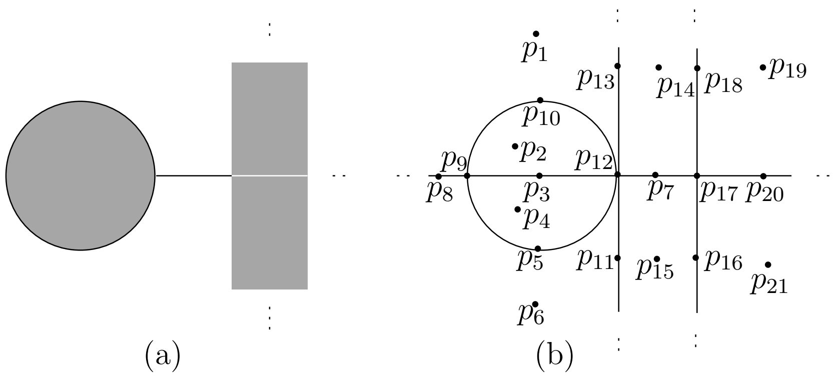

FIG. 2. *A semi-circular relation* (a) *and its carrier* (b).

where a, b, and c are real algebraic numbers with $a \neq 0$ or $b \neq 0$, and θ is either $\geq$ or $>$; or

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

where a, b, and c are real algebraic numbers with $c \neq 0$, and θ is either $\geq$ or $>$.

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 2(a) shows an example of a semi-circular relation. It is the set

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

Given such a semi-circular database, we consider all of the sets of the form $\{(x, y) \mid p(x, y) = 0\}$ for each polynomial $p(x, y)$ that occurs in the definition of the semi-circular relation.

For the semi-circular relation of Figure 2(a), these sets are shown in Figure 2(b) and 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 2(b) partitions the plane $\mathbf{R}^2$ into 21 classes, each of which belongs either entirely to the semi-circular relation or to its complement. In general, these partition classes can be disconnected. We then pick a representative of each of these classes, illustrated by points $p_1, \ldots, p_{21}$ in Figure 2(b). We can then represent a semi-circular relation R by a finite point database[1] that consists of three relations, L, P, and C, as follows:

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

The sets L and C are binary relations of points in the plane, while the set P is a unary relation of points.

[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 [38].

We 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, for example, because there may be more than one constraint formula describing it.

For the semi-circular database of Figure 2(a) we can have 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 the single tuple (p_3, p_5); and P consists of the points p_2, p_3, p_4, p_5, p_7, p_9, p_{10}, p_{12}, p_{19}, and p_{21}.

The complete plane $\mathbf{R}^2$ can be represented by $L = \emptyset$, $C = \emptyset$, and $P = \{p\}$ for any point p in $\mathbf{R}^2$.

We remark that an intensional *LPC*-representation of a semi-circular relation is lossless in the sense that the semi-circular relation can be reconstructed from the representation. In section 7, we show how to compute in $\mathsf{FO} + \mathsf{poly}$ an *LPC*-representation of a semi-circular relation and how to reconstruct, also in $\mathsf{FO} + \mathsf{poly}$, the semi-circular relation from its *LPC*-representation.

As an immediate consequence of the existence of quantifier-elimination algorithms for the real closed field, we get the following property.

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

4. The language EuPL.

We now define our first 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, but we are interested only in first-order database query languages. We therefore first explore the consequences of defining a Euclidean programming language without iteration. An additional feature of EuPL is that it includes a nondeterministic choice operator. We decided to include this operator as it is used frequently in the Euclidean constructions that we want to model, but we shall show that this choice operator is redundant, since, under appropriate assumptions, it can be simulated by other operations.

EuPL has one basic type $\langle \text{var} \rangle$ which ranges over points in the Euclidean plane. We use p, $q, \ldots$ to denote variables. There is no basic notion of lines and circles, since 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.[2]

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 [13]). The main reason that we have chosen to use a point representation in EuPL is to make the language similar to the database languages in the following sections, where such an encoding really is necessary. As far as EuPL is concerned, however, we could have defined a similar language with lines and circles as primitive notions—all of the results in the current section, apart from Theorem 4.2, would still hold.

The basic predicates in EuPL are as follows:
1. **defined**(p),
2. $p_1 = p_2$,
3. (i) p_1 **is on line** (p_2, p_3),

[2]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, or in other ways. Our choice is arbitrary, but tends to make actual constructions simpler.

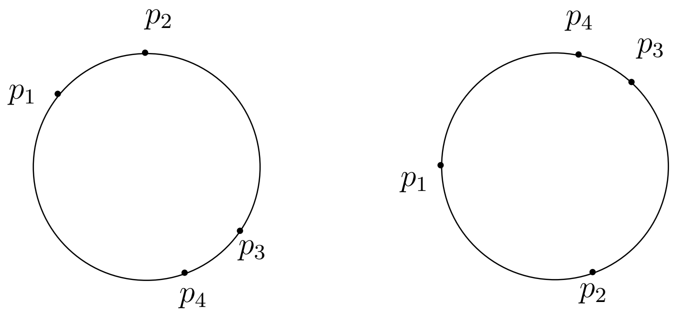

FIG. 3. *The two orientations for* **c-order**(p_1, p_2, p_3, p_4).

(ii) p_1 **is on circle** (p_2, p_3, p_4),
(iii) p_1 **is in circle** (p_2, p_3, p_4),
(iv) p_1 **is on the same side as** p_2 **of line** (p_3, p_4),

4. (i) **l-order**(p_1, p_2, p_3),
   (ii) **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, such as parallel lines.

Given our encoding of lines and circles, the meaning of the predicates in 3(a–d) should be clear. For example, p_1 **is in circle** (p_2, p_3, p_4) means that p_1 is in the circle with center p_2 and radius $d(p_3, p_4)$. The predicates in 4(a), 4(b) 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, in either the clockwise or the counterclockwise direction (see Figure 3). 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.[3] In the case of the intersection of two parallel lines (identical or not), q is undefined, and similarly for the intersection of two disjoint circles, or for the intersection of a disjoint circle and line. In the case of a line tangent to a circle or two circles that meet in a single point, q_1 and q_2 will be identical.

The choice operator, whose syntax is

$$\textbf{choose}\ p\ \textbf{such that}\ \langle\text{condition}\rangle\,,$$

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

[3]This actually introduces an additional nondeterminism into the language. This can be handled by straightforward modifications of the proofs that follow and will be ignored from now on.

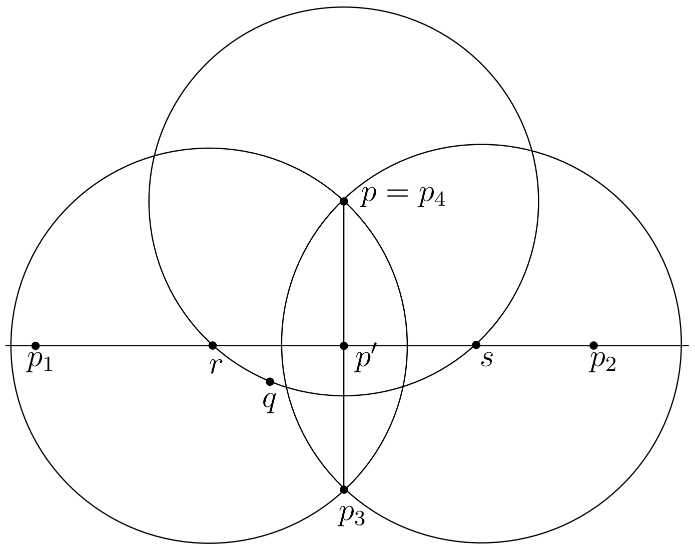

FIG. 4. *Construction of the perpendicular.*

The language also has a conditional statement **if** C **then** S_1 **else** S_2 with the usual semantics. A formal specification of the language appears in the appendix. The basic notion is that of a *multifunction* with n input variables and m output variables, representing points in the plane. Each multifunction is defined by a sequence of assignment statements and conditional statements, without looping, and its 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 4).

```
**multifunction** 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**
```

For another 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 that the ellipse itself is not constructible with ruler and compass and therefore, as we shall see in Theorem 4.3, cannot be defined by an EuPL program.

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 5):

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

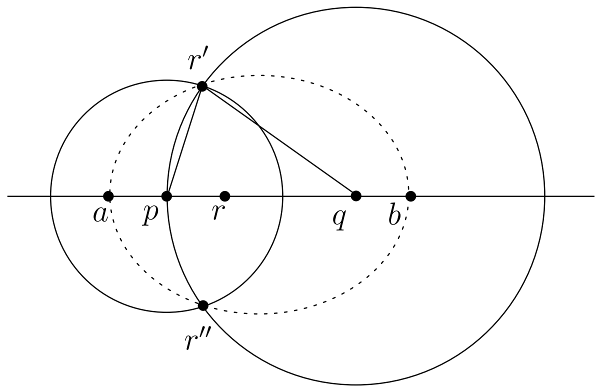

FIG. 5. *Construction of a point on an ellipse.*

```
    $r', r'' :=$ **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. Consequences of quantifier elimination: Representation independence. As pointed out above, EuPL is at least as powerful as a language with lines and circles as primitives. But EuPL is actually more powerful than desired. For example, if we are given a line represented by points (p_1, p_2), the EuPL program that returns p_1 would have no natural geometric interpretation.

This leads to the following problem: Given an EuPL program, does the result depend on the representation of the input lines and circles or not? The inputs and outputs of an EuPL program are just points, with no indication as to what they "really" represent. We therefore first need to 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, or circles) are represented by 1, 2, or 3 points, respectively, all we need are two equivalence relations on the input and output.

DEFINITION 4.1. *An* interpretation *of an* EuPL *program P is a pair of equivalence relations E_i and E_o on the input and 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 it is unlikely to have an intuitive result or be representation-independent. This means that we do not need to address the issue of an interpretation of the internal variables of a program.

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

First, we define representation dependence.

Definition 4.2. *Let P be an* EuPL *program with input parameters $(p_1, \dots, p_n)$ and output parameters $(q_1, \dots, q_m)$ for which an interpretation is fixed. We call P* representation-independent *if for any two inputs values $(a_1, \dots, a_n)$ and $(a'_1, \dots, a'_n)$ that represent the same points, lines, and circles in the plane (taking into account the fixed interpretation), P returns outputs $(b_1, \dots, b_m)$ and $(b'_1, \dots, b'_m)$, respectively, that also represent the same points, lines, and circles in the plane (taking into account the fixed interpretation).*

The following theorem follows from the fact that the input-output transformations in EuPL can be expressed in first-order logic over the reals and from the fact that the truth of sentences in this logic can be decided (via quantifier elimination) [38].

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

Proof. Let P be an EuPL multifunction, with inputs $\vec{\mathbf{p}} = (p_1, \dots, p_n)$ and outputs $\vec{\mathbf{q}} = (q_1, \dots, q_m)$. 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 $(p^x - q^x)^2 + (p^y - q^y)^2 = t^2 \wedge t \geq 0$, 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 of 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 what follows, whenever ψ_S is not defined explicitly, it will be a tautology. The separation between ϕ and ψ is not really needed for the current theorem, but will be used later in Theorem 4.2. The basic idea is that for each intersection operation we add a conjunct that says when the corresponding objects are defined and have an intersection, while for each choice operation we add a conjunct that says that the condition is satisfiable. The construction of ϕ and ψ is by induction on the rules defining P. We omit some of the straightforward cases. As a first step, rename variables if needed, to ensure they are not assigned a value more than once.

1. P is the sequence of statements $S_1; S_2; \dots; S_k$. ϕ_P is defined as $\phi_{S_1} \wedge \dots \wedge \phi_{S_k}$ and ψ_P as $\psi_{S_1} \wedge \dots \wedge \psi_{S_k}$.
2. Assignment statements:
   (i) S is $q :=$ **l-l-crossing**(p_1, p_2, p_3, p_4). ϕ_S can be informally given as

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

   Note that this formula is unsatisfiable whenever q is undefined.

   (ii) S is $q,\ q' :=$ **l-c-crossing**$(p_1, p_2, p_3, p_4, p_5)$. Let $\phi'_S(q, p_1, p_2, p_3, p_4, p_5)$ be

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

   which means that q is one of the intersection points. Then ϕ_S is

$$\begin{aligned}&\phi'_S(q, p_1, p_2, p_3, p_4, p_5) \wedge \phi'_S(q', p_1, p_2, p_3, p_4, p_5)\\ &\qquad \wedge (q \neq q' \vee (q = q' \wedge \neg \exists q'' (\phi'_S(q'', p_1, p_2, p_3, p_4, p_5) \wedge q'' \neq q)));\end{aligned}$$

   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.

   (iii) S is $q,\ q' :=$ **c-c-crossing**$(p_1, p_2, p_3, p_4, p_5, p_6)$. This is similar to the previous case.

3. S is the conditional

$$\textbf{if } C \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ end}.$$

ϕ_S is

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

and ψ_S is

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

4. S is the choice statement

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

ϕ_S is a tautology, and 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. C is **defined** $(\langle \text{var} \rangle)$. ϕ_C is defined as the appropriate Boolean value.

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 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 output of P is then independent of the representation if and only if the formula

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

holds in the real numbers.

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

Using the formulas ϕ_P and ψ_P, given in the proof of Theorem 4.1, it is clear that we can write an $\mathsf{FO} + \mathsf{poly}$-sentence that expresses that two programs have the same input-output behavior. Therefore, we have the following corollary.

COROLLARY 4.1. *Equivalence of* EuPL *programs is decidable.*

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 discussed below, "good" programs should be choice-independent anyway, so that the distinction is not very important.

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.

The following result holds both for interpreted and for uninterpreted EuPL programs.

COROLLARY 4.2. *It is decidable whether the result of an* EuPL *program depends on the choices made by* **choose** *operators.*

COROLLARY 4.3. *It is decidable whether the result of an interpreted* EuPL *program depends on the representation of its inputs and on the results of the choice operations.*

We now show that given two fixed 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.

We now consider a variant EuPL^{2c} of EuPL, which is just EuPL with two additional distinct constant points p_0 and p_0'.

THEOREM 4.2. *Every representation- and choice-independent* EuPL^{2c} *program* P *is equivalent to a program* P' *which does not use the* **choose** *operator.*

Proof. Let P be an EuPL^{2c} 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^{2c} statements, among which there is no choice statement, provided that the result is a point p satisfying $\psi(p, p_1, \ldots, p_n)$. We shall now construct such a sequence.

Note first that $\psi(p, p_1, \ldots, p_n)$ is a Boolean combination of basic choice predicates in EuPL^{2c}. We now show that there exists a formula $\psi'(p, p_1, \ldots, p_n)$, equivalent to $\psi(p, p_1, \ldots, p_n)$, such that ψ' is a Boolean combination of atomic predicates in all of which the variable p is the first variable. It will then follow that p belongs to an equivalence class of the plane determined by the lines and circles in these atomic predicates, as these predicates describe how p is located with respect to certain lines and circles determined by the points $p_1, \ldots, p_n$.

We show how to move the variable p to the first position for one specific case; the treatment of the other cases is similar. Consider the predicate

$$p_1\ \textbf{is on the same side as}\ p_2\ \textbf{of line}\ (p, p_3).$$

For points p, p_1, p_2, and p_3 that form a quadrangle, this expression is equivalent to $\neg((\varphi_1 \wedge \varphi_2) \vee (\neg\varphi_1 \wedge \neg\varphi_2))$, where φ_1 and φ_2 are, respectively,

$$p\ \textbf{is on the same side as}\ p_1\ \textbf{of line}\ (p_2, p_3)$$

and

$$p \textbf{ is on the same side as } p_2 \textbf{ of line } (p_1, p_3).$$

We now define an EuPL^{2c} program that produces a set $S_{\psi'}$ containing at least one representative point of each of the equivalence classes of the plane determined by the lines and circles in the formula $\psi'(p, p_1, \ldots, p_n)$. We then replace

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

by a formula that computes these representative points, checks for each of them whether the condition $\psi'(p, p_1, \ldots, p_n)$ holds, and returns the first such point.

To start with, let $S_{\psi'}$ be the set $\{p_1, \ldots, p_n\}$. Then for each predicate **c-order**(p, p_i, p_j, p_k) in ψ', construct the center of the circle through p_i, p_j, and p_k, and add it to $S_{\psi'}$. For each circle appearing in ψ' such that no point of $S_{\psi'}$ occurs in the circle, take the intersections of the circle and the line that connects the center to the fixed points p_0 or p'_0, and them to $S_{\psi'}$. Next, construct all of the intersection points of the circles and lines in the formula and add them to $S_{\psi'}$. This deals with all equivalence classes that are single points.

Next, for every pair of points in $S_{\psi'}$ add their midpoints to $S_{\psi'}$; for all pairs of points on a circle then add the midpoints of the arc segments between them, and for each unbounded line segment add the intersection of this segment and a circle whose center is the start of the segment and whose radius is the distance between p_0 and p'_0. This deals with the one-dimensional equivalence classes.

Finally, for all triples of points in $S_{\psi'}$, add their centroids to $S_{\psi'}$ This deals with the two-dimensional equivalence classes and completes the proof. $\square$

All of the languages that we shall discuss from now on are 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 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.

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 4.3. *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.

Example 4.3. Let p, q, r, and s be four different points as in Figure 6. Consider the following construction: First we 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 we 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 6).

The computation of this point requires iteration, as stated in the following lemma.

LEMMA 4.1. *The above construction cannot be expressed by an* EuPL *program.*

Proof. From the proof of Theorem 4.1 it follows that any EuPL program can be expressed in $\mathsf{FO} + \mathsf{poly}$. If the construction from Example 4.3 would be expressible

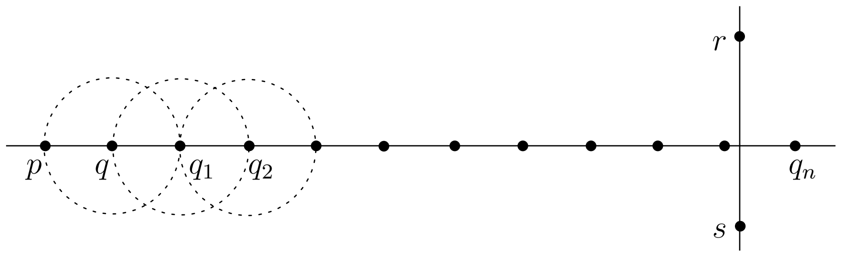

FIG. 6. *Non–first-order construction.*

by an EuPL-program, it would therefore be possible in FO + poly to test whether the distance from p to the vertical through r and s is a multiple of $d(p, q)$, which would allow integers to be definable in FO + poly. Results in [35] would imply the undecidability of FO + poly, a contradiction. ∎

5. The language EuQL. Our goal is to define a database query language for Euclidean geometry. In this section we describe an initial attempt, EuQL, at defining such a language. EuQL should be a declarative database language, so assignment statements are replaced by predicates. For example, the crossing-point operators become predicates rather than assignments. In addition, the **defined** predicate is not needed, as existential quantifiers can be used instead.

The relations of the input database are finite two-dimensional point relations, i.e., finite tuples of two-dimensional points, represented by real polynomial formulas. The relation R_i, of arity m_i, is an m_i-ary finite two-dimensional point relation, i.e., a $2m_i$-ary relation over the reals. An 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 logic 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** $(\langle\text{var}\rangle, \langle\text{var}\rangle, \langle\text{var}\rangle, \langle\text{var}\rangle, \langle\text{var}\rangle, \langle\text{var}\rangle)$,

EuQL has three constant points o, e_1, and e_2, with (o, e_1) perpendicular to (o, e_2). We shall refer to the line through o and e_1 as the *x-axis* and to the line through o and e_2 as the *y-axis*.

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 in the natural way.

For example, we have the following:

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.

The three special points are interpreted as a coordinate system.

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:

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

Unfortunately, it turns out that EuQL is too powerful. To show why, we define a query that constructs an ellipse, 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 $d(t_3, t_4)$:

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

THEOREM 5.1. EuQL *can express queries that are not constructible in Euclidean geometry.*

While this shows that EuQL does not match the intuition we had in mind, one might hope that it would still serve as a language between FO + lin and FO + poly. This is not the case, however, as the following result shows.

From Euclid, we know that multiplication can be performed with ruler and compass, and so the following theorem holds.

THEOREM 5.2. *On finite point databases,*[4] EuQL *has the same expressive power*[5] *as* FO + poly*.*

In order to obtain the desired language, we shall now restrict EuQL in an appropriate way.

[4]We define *finite point databases* as database instances over some database schema $R_1, \ldots, R_n$ in which the interpretation of each relation R_i is a finite set of points in $\mathbf{R}^2$.

[5]Let *can* be the canonical bijection mapping a point p of $\mathbf{R}^2$ to the pair (p^x, p^y) of its real coordinates. An EuQL query Q over an input schema $R_1, \ldots, R_n$ has the same expressive power as an FO + poly query Q' over the schema $R'_1, \ldots, R'_n$, where the arity of R'_i is double the arity of R_i if for any instance $A_1, \ldots, A_n$ over $R_1, \ldots, R_n$, $can(Q(A_1, \ldots, A_n)) = Q'(can(A_1), \ldots, can(A_n))$.

6. The language SafeEuQL.

In this section we define a subset of EuQL, called SafeEuQL, that consists of queries whose results, on finite databases, are constructible in Euclidean geometry. This subset consists of those EuQL queries that satisfy a syntactically defined *safety* condition, whose intuition is to restrict the domain over which variables range to be finite, as soon as the input database is finite.

A $\langle$disjunction$\rangle$ is defined as a disjunction of $\langle$conjunction$\rangle$'s, and a $\langle$conjunction$\rangle$ is a conjunction of $\langle$factor$\rangle$'s. A $\langle$factor$\rangle$ is a $\langle$term$\rangle$ or a $\neg\,\langle$term$\rangle$. Finally, a $\langle$term$\rangle$ is either $\exists\,\langle$var$\rangle\,(\langle$disjunction$\rangle)$ or is an EuQL primitive, including the three constant points. We say that an EuQL expression is in *safe-range normal form* if it can be defined as a $\langle$disjunction$\rangle$.

We now define the set of variables which are *safe* in an 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:

1. $\mathcal{S}v(R(v_1,\ldots,v_p))$ equals $\{v_1,\ldots,v_p\}$.
2. For each of the EuQL primitives φ, $\mathcal{S}v(\varphi)$ equals the empty set.
3. $\mathcal{S}v(\exists v\varphi)$ equals $\mathcal{S}v(\varphi) - \{v\}$.
4. $\mathcal{S}v(\neg\varphi)$ equals the empty set.
5. $\mathcal{S}v(\varphi_1 \wedge \varphi_2)$ equals the smallest set S, with respect to $\subseteq$, such that the following properties hold:
   (i) 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;
   (ii) 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;
   (iii) 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;
   (iv) 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
   (v) $\mathcal{S}v(\varphi_1) \cup \mathcal{S}v(\varphi_2)$ is a subset of S.

   All the above cases also hold for the appropriate variables when the remaining variables are constants. Showing existence of the set S is straightforward.
6. $\mathcal{S}v(\varphi_1 \vee \varphi_2)$ equals $\mathcal{S}v(\varphi_1) \cap \mathcal{S}v(\varphi_2)$.

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

1. *for each subformula of* φ *of the form* $\exists v\psi$*, it is the case that* $v \in \mathcal{S}v(\psi)$*, and*
2. *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 show that safety of an EuQL query is a 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 safe EuQL queries will be called SafeEuQL. The following closure property holds.

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 to B, every variable v in $\mathcal{S}v(\varphi)$ ranges over a finite domain.

First, let φ be quantifier-free. We prove the 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, observe that the only SafeEuQL expressions with v as a safe variable are those 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 holds.

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 $k + 1$. There are 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 to B, v ranges over finite domains, say D_1 and D_2. Then when φ is applied to B, v must range over a domain contained in $D_1 \cup D_2$.
2. $\varphi \equiv \psi_1 \wedge \psi_2$. Denote by S the union of $\mathcal{S}v(\psi_1)$ and $\mathcal{S}v(\psi_2)$. By the induction hypothesis, when ψ_1 and ψ_2 are applied to B, each variable of S ranges over a finite domain. Let D be the union of all these domains. Repeat the following process until v is in S. Consider every SafeEuQL primitive in φ which does not occur in any $\neg\psi$, where $\neg\psi$ is 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** $(v_2, v_3, v_4, v_5, v_6, v_7)$, where the points $v_2, \ldots, v_7$ are all in 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. Add the points in D' to D. The resulting set is still finite, and every variable of S ranges over at most the points in D. Since, by assumption, v is safe in φ, it follows that this process terminates after a finite number of steps. Thus, for φ applied on B, v ranges over a finite set of points.

Note that the case $\varphi \equiv \neg\psi$ cannot occur because it has no safe variables.

Next, consider a SafeEuQL term of the form $\exists v \psi$ with ψ quantifier-free. By the definition of safety, v must 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. Replace the formula $\exists v \psi$ in φ 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 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, observe that every EuQL primitive can be simulated with ruler and compass. Since every variable in a SafeEuQL expression applied to a finite point database ranges over a finite set of points, there exists a finite sequence of ruler-and-compass constructions which yields the same set of points as the SafeEuQL expression. Thus, for every SafeEuQL expression, the finite output database can be constructed with ruler and compass from the input database, which concludes the proof. ◻

THEOREM 6.2. *SafeEuQL has full arithmetical power on the coordinates of safe variables; i.e., we can subtract, add, multiply, and divide coordinates of such variables.*

Proof. Assume that p and q are safe variables. Using the three fixed points as a coordinate system, we write SafeEuQL queries 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. Without loss of generality, we can therefore assume that p and q are safe variables with coordinates of the form $(p_1, 0)$ and $(q_1, 0)$. If we then consider the well-known ruler-and-compass constructions for multiplication and division, it is easy to see that they can be expressed as SafeEuQL queries. This concludes the proof. ◻

7. The main results.

We now 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 $\mathsf{FO} + \mathsf{poly}$ expressible queries that map semi-circular relations to semi-circular relations.

To define these languages, we lift the query language SafeEuQL, which is defined on finite point databases, to a language called SafeEuQL$^{\uparrow}$, which is defined on semi-circular databases. This is done by interpreting these SafeEuQL$^{\uparrow}$ queries to work on the intensional representations of semi-circular databases defined in section 3.

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

Given an LPC-database 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 constant points o, e_1, and e_2, by constructing 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).

In the following, we shall not distinguish between the point and coordinate representation of an LPC-database; i.e., given L, P, and C relations, we will interpret them as points or coordinates depending on the context in which they are used.

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

Before defining SafeEuQL$^{\uparrow}$, we need two lemmas. The first is straightforward.

LEMMA 7.1. *There exists an $\mathsf{FO} + \mathsf{poly}$ 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.*

LEMMA 7.2. *There exists an $\mathsf{FO} + \mathsf{poly}$ 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.

$$\begin{tikzcd}
R_{\text{circ}} \arrow[r] \arrow[d, "\psi_{R_{\text{circ}}\to(L,P,C)}"'] & R_{\text{circ}} \\
(L,P,C) \arrow[r, "\mathsf{SafeEuQL}"] & (L,P,C) \arrow[u, "\psi_{(L,P,C)\to R_{\text{circ}}}"']
\end{tikzcd}$$

FIG. 7. *The query language* $\mathsf{SafeEuQL}^{\uparrow}$ *is closed on semi-circular relations.*

Proof. Let S be a semi-circular set. It is well known that the topological boundary of S, ∂S can be expressed in $\mathsf{FO}+\mathsf{poly}$ (e.g., using the first-order definition of ε-environments of points). The same is obviously true for the complement of S, S^c and therefore also for the boundary of the complement of S, ∂S^c.

Consider the following sets. Let L_S be the set of all triples (a,b,c) of $\mathbf{R}^3$ such that the line $ax+by+c=0$ has infinitely many points in common with ∂S or with ∂S^c. Let C_S be the set of all triples (a,b,r) of $\mathbf{R}^3$ such that the circle $(x-a)^2+(y-b)^2-r^2=0$ has infinitely many points in common with ∂S or with ∂S^c. Next, let us denote by $\partial^2 S$ the set consisting of all isolated points of ∂S and of ∂S^c and of all end points of half-lines, line segments, and circle segments on ∂S or ∂S^c (a point p is said to be an *end point* of a line segment l with carrier c if there exists an $\varepsilon>0$ and a point q such that $d(p,q)<\varepsilon$ and $q\in c\setminus l$; and end point of a circle segment is defined in a similar way). Let I_S be the set of all triples $(1,0,-a)$ and $(0,1,-b)$ of $\mathbf{R}^3$ such that (a,b) are the coordinates of a point of $\partial^2 S$.

It is clear that all lines and circles in a carrier of S are appearing in $L_S\cup I_S$, respectively C_S, albeit multiple times (except for the lines given by I_S). We can consider the first-order definable equivalence relation $\sim_L$ on $L_S\cup I_S$, defined as $(a,b,c)\sim_L(a',b',c')$ if and only if the equations $ax+by+c=0$ and $a'x+b'y+c'=0$ define the same line (i.e., if and only if $ac'=a'c$ and $bc'=b'c$). We can also consider the first-order definable equivalence relation $\sim_C$ on C_S, defined as $(a,b,r)\sim_L(a',b',r')$ if and only if the equations $(x-a)^2+(y-b)^2-r^2=0$ and $(x-a')^2+(y-b')^2-r'^2=0$ define the same circle (i.e., if and only if $a=a'$, $b=b'$, and $r=\pm r'$). By the definable choice property (see, e.g., Property 1.2 in Chapter 6 of [39]), representatives of each equivalence class can be first-order defined. Once this is done it is easy to obtain two representative points on each line in the L relation of S and the center and a representative point on each circle in the C relation of S.

It remains to be shown that also the P relation of S can be first-order defined. The sets $L_S\cup I_S$ and C_S partition $\mathbf{R}^2$ according to the following first-order definable equivalence relation $\sim_S$: $(x,y)\sim_S(x',y')$ if and only if for all $(a,b,c)\in L_S\cup I_S$, $ax+by+c$ and $ax'+by'+c$ have the same *sign* ($=0$, <0, or >0) and for all $(a,b,r)\in C_S$, $(x-a)^2+(y-b)^2-r^2$ and $(x'-a)^2+(y'-b)^2-r^2$ have the same sign.

Since $\sim_S$ is first-order definable, again by the definable choice property, representatives of each equivalence class of $\sim_S$ can be first-order defined and added to the P relation of S whenever these representatives belong to S. $\square$

DEFINITION 7.1. $\mathsf{SafeEuQL}^{\uparrow}$ *is the set of all queries Q of the form*

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

where Q_{SafeEuQL} *is a* $\mathsf{SafeEuQL}$ *query (see Figure 7).*

$$\begin{tikzcd}
& {(L,P,C)} \arrow[rr, "Q_{\mathsf{SafeEuQL}}"] & & {(L,P,C)} \arrow[dd, "Q_{(L,P,C)\to R_{\mathrm{lin}}}"'] \\
{} \arrow[ur, "Q_{\mathsf{SafeEuQL}}"] \arrow[dr, "Q_{\mathrm{lin}}"'] & & & \\
& R_{\mathrm{lin}} \arrow[uu, "Q_{R_{\mathrm{lin}}\to(L,P,C)}"'] \arrow[rr, "Q_{\mathrm{lin}}"] & & R_{\mathrm{lin}}
\end{tikzcd}$$

FIG. 8. *Any* $\mathsf{FO}+\mathsf{lin}$ *query on semi-linear relations can be simulated in* $\mathsf{SafeEuQL}$ *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 LPC-database can be defined in* $\mathsf{SafeEuQL}$.

A $\mathsf{SafeEuQL}^{\uparrow}$ query is therefore a composition of three queries. First, the query 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 another LPC-database. Finally, the coordinate representation of this LPC-database is mapped to the semi-circular relation it represents. (A similar "lifting" idea is used in [5].)

The language $\mathsf{SafeEuQL}^{\uparrow}$ is closed on the class of semi-circular relations. This is illustrated in Figure 7. We thus have a syntactically defined subclass of $\mathsf{FO}+\mathsf{poly}$ that is closed on semi-circular relations.

7.2. On semi-linear relations the language $\mathsf{SafeEuQL}^{\uparrow}$ 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 $\mathsf{SafeEuQL}$ on the intensional level. Therefore, we can conclude that $\mathsf{SafeEuQL}^{\uparrow}$, on semi-linear databases, can express every $\mathsf{FO}+\mathsf{lin}$ query. On the other hand, it is clear that $\mathsf{SafeEuQL}^{\uparrow}$ is more expressive than $\mathsf{FO}+\mathsf{lin}$ on linear inputs, simply because the latter can output linear databases only, while the former can also produce semi-circular ones.

This is illustrated in Figure 8 and stated more precisely in the following theorem, whose proof follows later in this section.

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* $\mathsf{SafeEuQL}$ *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 main difficulty is to prove the existence of the $\mathsf{SafeEuQL}$ query which simulates a given $\mathsf{FO}+\mathsf{lin}$ query. From Lemmas 7.1 and 7.2, it follows that there exist $\mathsf{FO}+\mathsf{poly}$ 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 Lemmas 7.1 and 7.2 shows that the corresponding LPC-database for a linear relation has an empty C-relation; we shall refer to such an encoding as an LP-database.

The main difficulty in simulating an $\mathsf{FO} + \mathsf{lin}$ expression by a $\mathsf{SafeEuQL}$ expression is that, in general, subformulas of $\mathsf{FO} + \mathsf{lin}$ expressions operate in a higher dimensional space, due to the quantifiers that may be present in the expression. Therefore, the LP-representation technique for two-dimensional linear relations has to be generalized 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 topologically open, convex cells, such that a finite number of these cells constitute the given semi-linear set. These $(n-1)$-dimensional hyperplanes are finitely represented with n linearly independent points. An n-dimensional point p can be represented within $\mathsf{SafeEuQL}$ as a tuple of n two-dimensional points as $((p_1, 0), \ldots, (p_n, 0))$, where p_i is the ith coordinate of p. Based on this, each n-dimensional semi-linear set is represented in $\mathsf{SafeEuQL}$ by a pair of relations (H^n, P^n), where H^n is a $2n^2$-ary relation containing the representation of a finite number of $(n-1)$-dimensional hyperplanes, and where P^n is a $2n$-ary relation containing the representations of the representatives of the partition classes that constitute the semi-linear set. More precisely, the hyperplane in $\mathbf{R}^n$ through the points $\overline{p_i} = (p_{i,1}, \ldots, p_{i,n}) \in \mathbf{R}^n$, $1 \leq i \leq n$, is stored in H^n by the $2n^2$-ary tuple $(((p_{1,1}, 0) \ldots, (p_{1,n}, 0)), \ldots, ((p_{n,1}, 0) \ldots, (p_{n,n}, 0)))$. As an example, the semi-linear subset $\{(x, y, z) \in \mathbf{R}^3 \mid z > 0 \wedge y > 0\}$ of $\mathbf{R}^3$ could be given by (H^3, P^3), with $H^3 = \{((0,0), (0,0), (0,0), (1,0), (0,0), (0,0), (0,0), (1,0), (0,0), (0,0), (0,0), (0,0), (1,0), (0,0), (0,0), (0,0), (0,0), (1,0))\}$ and $P_3 = \{((1,0), (1,0), (1,0))\}$.

Before giving the proof of Theorem 7.1, we prove two lemmas.

The first follows immediately from the fact that we can do arithmetic with Euclidean constructions.

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. There exists a* $\mathsf{SafeEuQL}$ *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.*

LEMMA 7.4. *Denote by 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* $\mathsf{SafeEuQL}$ *expression which computes the relation P^n that contains at least one representative point for every partition class induced by the hyperplanes of H^n.*

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. Therefore, a finite set of representatives of this new partition is also a set of representatives of the old partition.

Next, take n hyperplanes from the relation H^n which are linearly independent, i.e., if no pair is either parallel or equal. We can test this in $\mathsf{SafeEuQL}$ as follows. Let p be an arbitrary point on the first hyperplane and q on the second. If, for each point r on the first hyperplane, the point as $r + q - p$ belongs to the second hyperplane, the two hyperplanes are equal or parallel. Since p, q, and r have to be in H^n, they must be safe variables, and by Theorem 6.2 we obtain a $\mathsf{SafeEuQL}$ expression which computes $r + q - p$. From Lemma 7.3 we can test in $\mathsf{SafeEuQL}$ whether a point belongs to a given hyperplane, and so we can test linear independence of hyperplanes in $\mathsf{SafeEuQL}$.

Every set of n linearly independent hyperplanes of the n-dimensional space intersects in exactly one point. Using Theorem 6.2 again, we construct a $\mathsf{SafeEuQL}$ expression for computing this intersection point. Denote by I the set of all of intersection points of all sets of n linearly independent hyperplanes of H^n. (I cannot be empty since each hyperplane of H^n intersects at least $n-1$ coordinate planes and therefore contributes at least one point to I.)

We now compute representative points of the *bounded* partition classes 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 *corner points*, which must be in I, and the barycenter of these corner points can be taken as a representative, which can be expressed in $\mathsf{SafeEuQL}$.

For unbounded partition classes, the set I may not suffice to compute the representative points. To handle this case we use a "bounding box": Each partition class will have a nonempty intersection with this bounding box, and we can choose a representative of the intersection of the partition class with the bounding box.

We now show how to construct this bounding box. For each coordinate plane of the n-dimensional space, we compute, in $\mathsf{SafeEuQL}$, two hyperplanes parallel with this coordinate plane such that all points of I are between these hyperplanes.

Denote by H_B the resulting set of $2n$, and let B be the open n-dimensional bounding box defined by the hyperplanes of H_B. We claim that B has a nonempty 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 was obtained from intersections of hyperplanes of H^n and is therefore contained in B. Since B is open, there exists a neighborhood of the corner point which is completely contained within B. The corner point, however, is also in the topological closure of its partition class, and therefore, this neighborhood has a nonempty intersection with the partition class. Therefore B has a nonempty 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 linearly independent hyperplanes of $H^n \cup H_B$, which can be computed in $\mathsf{SafeEuQL}$. 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 nonempty intersection with B, the set P^n contains a representative for each partition class induced by the hyperplanes of H^n. $\square$

Proof of Theorem 7.1. Assume that 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 relation, and let O_{lin} be the result of Q_{lin} applied on I_{lin}.

Denote the LP-representations for I_{lin} and O_{lin} by I_L, I_P and O_L, O_P, respectively, which can be computed in $\mathsf{FO}+\mathsf{poly}$ (see Lemma 7.2). We now construct $\mathsf{SafeEuQL}$ 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 $Q_P(I_L, I_P) = O_P$, where O_L and O_P are an LP-representation of the output $O_{lin} = Q_{lin}(I_{lin})$.

We prove this by induction on the structure of φ. For each subformula of φ with n free variables, we construct two $\mathsf{SafeEuQL}$ queries that construct the relations H^n and P^n, corresponding to the two parts of the LP-representation of the result.

1. *Atomic formula of the form* $I_{lin}(x, y)$. For each tuple (p, q) of I_L, H^2 should contain 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 p and q. For each tuple (p) of I_P, P_I should contain

a tuple $((p_x, 0), (p_y, 0))$. This can easily be expressed in SafeEuQL.

2. *Atomic formula of the form* $\sum_{i=1}^{n} a_i x_i \ \theta\ 0$, *with* $\theta \in \{=, <, >\}$. There exist n linearly independent points $p_1, \ldots, p_n$ such that the smallest affine space containing $p_1, \ldots, p_n$ is precisely the hyperplane given by $\sum_{i=1}^{n} a_i x_i = 0$, and there also exists a point p satisfying $\sum_{i=1}^{n} a_i x_i \ \theta\ 0$. Furthermore, the coordinates of these points can be computed in SafeEuQL. From this, it follows immediately that H^n and P^n can be expressed in SafeEuQL.
3. $\varphi_1(x_1, \ldots, x_m) \vee \varphi_2(x_1, \ldots, x_n)$. If $m \neq n$, assume without loss of generality that $m < n$. Assume that we have already computed the sets (H_1^m, P_1^m) and (H_2^m, P_2^m) in SafeEuQL. We first convert (H_1^n, P_1^n) to a representation of the formula $\varphi_1(x_1, \ldots, x_m, \ldots, x_n)$ in n-dimensional space by padding the representation of each point with $n - m$ zeros.
The representation (H^n, P^n) of $\varphi_1(x_1, \ldots, x_n) \vee \varphi_2(x_1, \ldots, x_n)$ is then computed as follows. H^n is the union of H_1^n and H_2^n. Let P be a set of representatives of all the partition cells induced by the hyperplanes represented by H^n. The set P^n is then obtained from P as
$$\{x \mid P(x) \wedge \exists y((P_1^m(y) \wedge \mathit{SameSide}(H_1^m; x, y)) \vee (P_2^m(y) \wedge \mathit{SameSide}(H_2^m; x, y)))\}\ .$$
4. $\neg\varphi_1(x_1, \ldots, x_m)$. In this case, $H^m = H_1^m$ and $P^m = \{x \mid P(x) \wedge \neg\exists y(P_1^m(y) \wedge \mathit{SameSide}(H_1^m; x, y))\}$.
5. $\exists x_i \varphi_1(x_1, \ldots, x_m)$. Let H^m and P^m be the representation of φ_1. For every two hyperplanes of H^m, compute a finite representation of their intersection, and project this representation onto the appropriate $m-1$ dimensions. If the projection of two points coincides, introduce an arbitrary new point, so that we obtain $(m-1)$ linearly independent points denoting a hyperplane in the ith coordinate plane, and add a tuple with these $(m-1)$ points to H^{m-1}. To compute P^{m-1}, let P be the set of all representatives of the partition induced by the hyperplanes in H^{m-1}. Let p be a point of P and q a point of P^m. Compute the intersection point r of the perpendicular to the ith coordinate plane through p with the hyperplane through q parallel with the ith coordinate plane. If q and r belong to the same partition class induced by the hyperplanes of H^m, add p to P^{m-1}. It is straightforward to verify that this can be computed in SafeEuQL.

We have obtained two SafeEuQL queries that compute the relations
$$\{((p_1, 0), (p_2, 0), (q_1, 0), (q_2, 0)) \mid O_L((p_1, p_2), (q_1, q_2))\}$$
and
$$\{((p_1, 0), (p_2, 0)) \mid O_P((p_1, p_2))\}.$$
From these, computing O_L and O_P is trivial. □

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

We 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 for the formulation of "nonconstructible" queries and therefore is more powerful than SafeEuQL$^{\uparrow}$.

DEFINITION 7.2. *Let* FO + poly$_{\text{circ}}$ *be the set of* FO + poly *queries that map semi-circular relations to semi-circular relations.*

The following result follows immediately from Lemma 7.5 below.

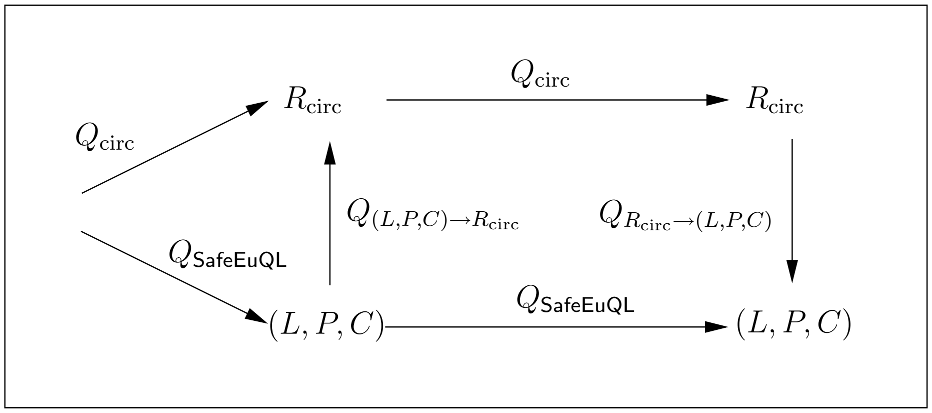
$$\begin{tikzcd}
 & R_{\text{circ}} \arrow[rr, "Q_{\text{circ}}"] & & R_{\text{circ}} \arrow[d, "Q_{R_{\text{circ}}\to(L,P,C)}"'] \\
\cdot \arrow[ur, "Q_{\text{circ}}"] \arrow[dr, "Q_{\mathsf{SafeEuQL}}"'] & & & \\
 & (L,P,C) \arrow[uu, "Q_{(L,P,C)\to R_{\text{circ}}}"'] \arrow[rr, "Q_{\mathsf{SafeEuQL}}"] & & (L,P,C)
\end{tikzcd}$$

FIG. 9. *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.*

THEOREM 7.2. $\mathsf{SafeEuQL}^{\uparrow}$ *is a strict subset of* $\mathsf{FO}+\mathsf{poly}_{\text{circ}}$.

LEMMA 7.5 (Figure 9). *For every* $\mathsf{SafeEuQL}$ *query, 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 $\mathsf{FO}+\mathsf{poly}$. The same holds for $\mathsf{SafeEuQL}$, since it is a sublanguage of EuQL. Let $\tilde{Q}_{\text{circ}}$ be the $\mathsf{FO}+\mathsf{poly}$ query which simulates the $\mathsf{SafeEuQL}$ 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 above conditions.

For the second part, consider the query that maps a semi-circular relation consisting of a line segment qr and a point p that is not collinear with q and r to the same 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 $\mathsf{FO}+\mathsf{poly}$. Since the query maps every semi-circular relation to a semi-circular relation, it belongs to $\mathsf{FO}+\mathsf{poly}_{\text{circ}}$. However, it is not expressible in $\mathsf{SafeEuQL}^{\uparrow}$, 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 $\mathsf{FO}+\mathsf{poly}_{\text{circ}}$ have the same expressive power.

8. Conclusion.

8. Conclusion. Figure 10 summarizes our results.

1. On the bottom level of Figure 10, we have $\mathsf{FO}+\mathsf{lin}$ as a query language on semi-linear relations. Recall that queries concerning Euclidean distance are not expressible in this language. Not only does the data model only

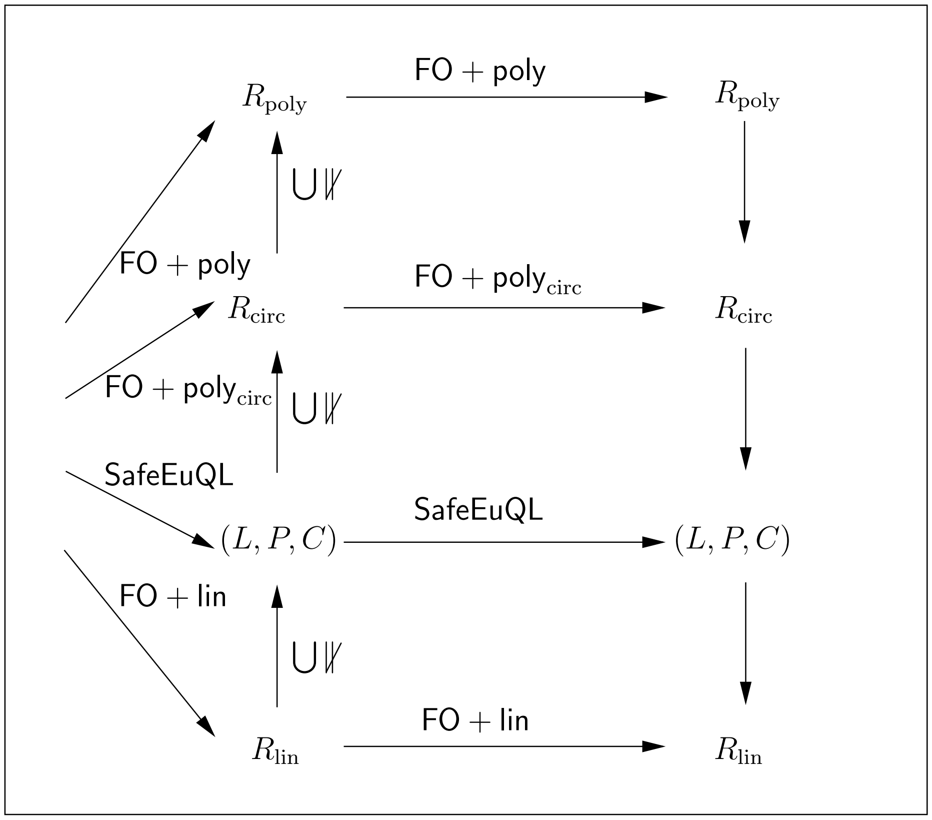

FIG. 10. *Comparison of the different query languages.*

allow semi-linear relations, but, moreover, there are FO + poly queries mapping semi-linear relations to semi-linear relations that are not expressible in FO + lin, for example, the transformation of a relation into its convex hull [41].

2. 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 larger class of relations than the semi-linear ones. All queries expressible in SafeEuQL are constructible by ruler and compass. So, the trisection of a given angle, for instance, is *not* expressible in SafeEuQL.
3. In FO + 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 FO + poly$_{\mathrm{circ}}$ has the same expressive power as FO + poly on semi-circular relations.
4. On the top level, we have FO + 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 FO + poly$_{\mathrm{circ}}$.

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 than 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⟩)∗
```

REFERENCES

[1] D. Abel and B. C. Ooi, eds., *Proceedings of the Third International Symposium on Spatial Databases*, Lecture Notes in Comput. Sci. 692, Springer-Verlag, Berlin, 1993.

[2] F. Afrati, T. Andronikos, and T. Kavalieros, *On the expressiveness of first-order constraint languages*, in Proceedings of the First Workshop on Constraint Databases and Their Applications, Lecture Notes in Comput. Sci. 1034, G. Kuper and M. Wallace, eds., Springer-Verlag, Berlin, 1995, pp. 22–39.

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

[4] S. Basu, R. Pollack, and M.-F. Roy, *On the combinatorial and algebraic complexity of quantifier elimination*, J. ACM, 43 (1996), pp. 1002–1046.

[5] M. Benedikt and L. Libkin, *Safe constraint queries*, SIAM J. Comput., 29 (2000), pp. 1652–1682.

[6] J. Bochnak, M. Coste, and M.-F. Roy, *Real Algebraic Geometry*, Springer-Verlag, Berlin, 1998.

[7] A. Buchmann, ed., *Proceedings of the First International Symposium on Spatial Databases*, Lecture Notes in Comput. Sci. 409, Springer-Verlag, Berlin, 1989.

[8] B. F. Caviness and J. R. Johnson, eds., *Quantifier Elimination and Cylindrical Algebraic Decomposition*, Springer-Verlag, Wien, New York, 1998.

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

[10] M. J. Egenhofer and J. R. Herring, eds., *Proceedings of the Fourth International Symposium on Spatial Databases*, Lecture Notes in Comput. Sci. 951, Springer-Verlag, Berlin, 1995.

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

[12] E. Engeler, *Foundations of Mathematics*, Springer-Verlag, Berlin, 1992.

[13] H. Eves, *College Geometry*, Jones and Barlett, Boston, 1995.

[14] M. Giusti, J. Heintz, J. E. Morais, J. Morgenstern, and L. M. Pardo, *Straight-line programs in geometric elimination theory*, J. Pure Appl. Algebra, 124 (1998), pp. 101–146.

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

[16] S. Grumbach, P. Rigaux, M. Scholl, and L. Segoufin, *DEDALE, a spatial constraint database*, in Proceedings of the Sixth International Workshop on Database Programming Languages, Lecture Notes in Comput. Sci. 1369, Springer-Verlag, Berlin, 1998, pp. 124–135.

[17] S. Grumbach and J. Su, *Towards practical constraint databases*, in Proceedings of the 15th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, ACM Press, New York, 1996, pp. 28–39.

[18] S. Grumbach and J. Su, *Finitely representable databases*, J. Comput. System Sci., 55 (1997), pp. 273–298.

[19] S. Grumbach and J. Su, *Queries with arithmetical constraints*, Theoret. Comput. Sci., 173 (1997), pp. 151–181.

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

[21] O. Günther and H.-J. Schek, eds., *Proceedings of the Second International Symposium on Spatial Databases*, Lecture Notes in Comput. Sci. 525, Springer-Verlag, Berlin, 1991.

[22] R. H. Güting, ed., *Advances in Spatial Databases—6th International Symposium (SSD '99)*, Lecture Notes in Comput. Sci. 1651, Springer-Verlag, Berlin, 1999.

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

[24] T. Heath, *The Thirteen Books of Euclid's Elements*, Dover, New York, 1956.

[25] J. Heintz and B. Kuijpers, *Constraint databases, data structures and efficient query evaluation*, in Proceedings of the First International Symposium on Applications of Constraint Databases (CDB'04), Lecture Notes in Comput. Sci. 3074, B. Kuijpers and P. Revesz, eds., Springer-Verlag, Berlin, 2004, pp. 1–24.

[26] D. Hilbert, *Grundlagen der Geometrie*, Teubner, Leipzig, 1899.

[27] J. E. Hopcroft and J. D Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison–Wesley, Reading, MA, 1979.

[28] P. C. Kanellakis, G. M. Kuper, and P. Z. Revesz, *Constraint query languages*, in Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Nashville, TN, 1990, pp. 299–213.

[29] P. C. Kanellakis, G. M. Kuper, and P. Z. Revesz, *Constraint query languages*, J. Comput. System Sci., 51 (1995), pp. 26–52.

[30] G. Kuper, L. Libkin, and J. Paredaens, eds., *Constraint Databases*, Springer-Verlag, Berlin, 2000.

[31] J. L. Lassez, *Querying constraints*, in Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, ACM Press, New York, 1990, pp. 288–298.

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

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

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

[35] J. Robinson, *Definability and decision problems in arithmetic*, J. Symbolic Logic, 14 (1949), pp. 98–114.

[36] M.-F. Roy, S. Basu, and R. Pollack, *Algorithms in Real Algebraic Geometry*, Algorithms Comput. Math. 10, Springer-Verlag, Berlin, 2003.

[37] M. Scholl and A. Voisard, eds., *Proceedings of the Fifth International Symposium on Spatial Databases*, Lecture Notes in Comput. Sci. 1262, Springer-Verlag, Berlin, 1997.

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

[39] L. van den Dries, *Tame Topology and O-minimal Structures*, Cambridge University Press, Cambridge, UK, 1998.

[40] L. Vandeurzen, M. Gyssens, and D. Van Gucht, *On the desirability and limitations of linear spatial query languages*, in Proceedings of the Fourth International Symposium on Spatial Databases, Lecture Notes in Comput. Sci. 951, M. J. Egenhofer and J. R. Herring, eds., Springer-Verlag, Berlin, 1995, pp. 14–28.

[41] L. Vandeurzen, M. Gyssens, and D. Van Gucht, *On query languages for linear queries definable with polynomial constraints*, in Proceedings of the Second International Conference on Principles and Practice of Constraint Programming, Lecture Notes in Comput. Sci. 1118, E. C. Freuder, ed., Springer-Verlag, Berlin, 1996, pp. 468–481.

[42] M. Ziegler, *Einige unentscheidbare Körpertheorien*, Enseign. Math. (2), 28 (1982), pp. 269–280.