Generalized Relational Algebra: Modeling Spatial Queries in Constraint Databases*

A. Belussi [1] E. Bertino [2] M. Bertolotto [3] B. Catania [2]

[1] Dipartimento di Elettronica e Informazione
Politecnico di Milano
Piazza L. da Vinci 32
20133 Milano, Italy
e-mail: belussi@elet.polimi.it

[2] Dipartimento di Scienze dell'Informazione
Università degli Studi di Milano
Via Comelico 39/41
20135 Milano, Italy
e-mail: bertino@hermes.mc.dsi.unimi.it
e-mail: catania@ghost.dsi.unimi.it

[3] Dipartimento di Informatica e Scienze dell'Informazione
Università degli Studi di Genova
Via Dodecaneso 35
16146 Genova, Italy
e-mail: bertmic@disi.unige.it

Abstract. The introduction of constraints in database languages is an appealing direction. In this paper we present a model for constraint databases, derived from the one proposed in [13], and a family of generalized relational algebra languages, derived from the language proposed in [12] and in [19]. Each language of the family is obtained by specifying a certain logical theory for constraints and a particular application domain, by means of a set of functions. We then analyze the issues concerning spatial data modeling in the proposed constraint data model and the application of the proposed language to model spatial queries. Indeed, constraint databases offer a powerful formalism to express sets of points embedded in n-dimensional spaces.

1 Introduction

Constraint programming is a completely declarative paradigm by which computations are described by specifying how they are constrained. In data modeling, the main advantage of using constraints is that they serve as a unifying data type for the (conceptual) representation of heterogeneous data. In particular, the benefit of this approach is emphasized when complex knowledge (for

* The work reported in this paper was partially supported by CNR under grant number 95.00430.CT12 "Modelli e sistemi per il trattamento di dati ambientali e terrioriali".

example, spatial or temporal data) has to be combined with some descriptive non-structured information (such as names or figures).

During the last few years, a lot of work has been done in order to introduce constraints in databases, both as data and as language primitives. One of the first approaches is certainly the one proposed in [10]. There, constraints are inserted in the relational algebra in order to model infinite relations. However, this introduction is only at the language level. Constraints are not inserted into data.

The first general design principle to make this integration feasible has been proposed in [13], where a general framework for constraint query languages has been defined. The framework is based on the simple idea that a constraint can be seen as an extension of a relational tuple, or, viceversa, that a relational tuple can be interpreted as a simple form of constraint. The new constraint tuples are called *generalized*. In the same paper, a calculus for constraint databases is proposed and shown to be tractable from a computational point of view. An algebra equivalent to the calculus introduced in [13] has been proposed in [12].

All the subsequent work extends these results in some ways. In [2], linear constraints are inserted into an object-oriented framework. The resulting query language can be seen as an extension of XSQL [14] and it has been proved to be equivalent to the language obtained by inserting linear constraints in SQL. In [19], constraints are used in modeling spatial data and spatial queries. A theory of spatial queries has also been proposed, adapting the concept of computable query proposed in [4] to the spatial case.

Neither the algebra proposed in [12] and in [19], nor the calculus proposed in [13] are tailored to a specific application. They can be used whatever the applicative context, regardless of the meaning given to constraints. However, we claim that each application needs some particular procedures. These procedures may either be not expressible in the constraint language or require a specific implementation.

In order to consider this aspect within a relational context, we define a family of relational languages denoted GRA($\Phi, \mathcal{F}$) (*Generalized Relational Algebra* tailored on Φ and $\mathcal{F}$). Φ represents a decidable logical theory used to specify constraints, and $\mathcal{F}$ represents a set of recursive total functions, that we may think of as a set of functions needed in a particular application domain. Hence, for each choice of Φ and of $\mathcal{F}$, we obtain a distinct language in the family.

Each language of the family is composed of two sets of operators. *Tuple operators* act on generalized tuples by manipulating the set of relational tuples they represent. Thus, they do not consider generalized tuples as unique objects. Rather, they consider them as a collection of relational tuples. On the other hand, *set operators* consider generalized tuples unitarily as single objects, without the use of explicit tuple identifiers.

Note that when Φ is the linear polynomial constraints theory and $\mathcal{F}$ is empty, the definition of tuple operators exactly coincides with the algebra operators proposed in [12]. Set operators do not increase the expressive power of the algebra, though functions in $\mathcal{F}$ do increase. Thus, set operators represent a compact way

to formalize operations on generalized tuples as a whole, that may be useful from a user point of view.

In order to better analyze the power of the proposed operators, we tailor the GRA($\Phi, \mathcal{F}$) to a spatial application domain. We show that the set of spatial data types (point, line and polygon), usually adopted in spatial database systems, can be simply mapped to linear constraints. Moreover, by analyzing the various spatial data models proposed in the literature [5, 8, 9, 21, 22], we show how a large part of spatial queries can be mapped to GRA expressions.

The remainder of the paper is organized as follows. In Section 2, the generalized constraint model is introduced. In Section 3, the family of Generalized Relational Algebra languages is defined, discussing the meaning of all the new introduced operators. Then, a particular language of the family, which is tailored for spatial applications, is considered. To this purpose, in Section 4, a representation of some spatial data in terms of our generalized model is proposed. The considered logical theory is that of linear polynomial constraints. Then, in Section 5, we show how several spatial queries can be defined within such language. The analysis shows that the introduced set operators are very useful in defining such queries. Finally, Section 6 presents some conclusions and outlines future work.

2 The Generalized Relational Model

The model we consider is an extension of the model proposed in [13]. The considered extension can be seen as a simplification from a data representation point of view.

In the following, a *constraint* identifies a formula of a decidable logical theory. Several classes of constraints have been proposed. For example, the class of real polynomial constraints identifies formulas of the form $p(X_1, ..., X_n)\ \theta\ 0$, where p is a polynomial with real coefficients in variables $X_1, ..., X_n$ and $\theta \in \{=, \neq, \leq, <, \geq, >\}$.

The following definition introduces the model.

Definition 1. Let Φ be a class of constraints (a decidable logical theory).

- A *generalized tuple* over variables $x_1, ..., x_k$ in the logical theory Φ is a finite conjunction $\varphi_1 \wedge ... \wedge \varphi_N$, where each φ_i, $1 \leq i \leq N$, is a constraint in Φ. The variables in each φ_i are all free and among $x_1, ..., x_k$.
- A *disjunctive generalized tuple* (abbreviated as *d-generalized tuple*) over variables $x_1, ..., x_k$ in the logical theory Φ is a finite disjunction of generalized tuples over variables $x_1, ..., x_k$.
- A *generalized relation of arity k* in Φ is a finite set $r = \{\psi_1, ..., \psi_M\}$ where each ψ_i, $1 \leq i \leq M$, is a d-generalized tuple over variables $x_1, ..., x_k$ and in Φ.
- The *formula corresponding to a generalized relation* $r = \{\psi_1, ..., \psi_M\}$ is the disjunction $\psi_1 \vee ... \vee \psi_M$.
- A *generalized database* is a finite set of generalized relations. $\diamond$

Each d-generalized tuple identifies a set of relational tuples. For example, the generalized tuple $X < 2 \wedge Y = 3$ identifies all the relational tuples with two attributes, X and Y, having 3 as value for Y and a number greater than 2 as a value for X. The set of relational tuples represented by a d-generalized tuple t is denoted by $ext(t)$. A d-generalized tuple t is *inconsistent* if $ext(t) = \emptyset$.

In our model, a d-generalized tuple is a finite disjunction of generalized tuples. In the model proposed in [13], a generalized relation is a finite disjunction of generalized tuples. Thus, the d-generalized tuple introduced in Definition 1 can be seen as a compact form for representing a portion of a generalized relation, as it has been introduced in [13]. As we will see in Section 4, this could be very useful for modeling spatial concepts. Indeed, a generalized tuple can only represent a convex figure. For representing concave figures either a d-generalized tuple or a set of generalized tuples should be used. In the latter case, generalized tuples should have an additional variable, encoding an object identifier. This fact is better explained by the following example.

Example 1. Suppose R is a generalized relation, containing figures in the Euclidean plane. Assume the current instance of R contains the figures shown in Figure 1. It is simple to show that the concave object in Figure 1 can be decomposed in two convex figures, representing two rectangles. A possible representation in the generalized relational model of the considered objects is the following:

$$(a \leq X \leq c) \wedge (b \leq Y \leq d)$$

$$(e \leq X \leq f) \wedge (g \leq Y \leq h) \vee (f \leq X \leq l) \wedge (g \leq Y \leq i)$$

In the representation of the concave object, each disjoint represents a convex polygon belonging to the convex decomposition of the figure.

If we want to represent the instance of R by using generalized tuples, the schema of R must contain an additional variable, encoding the figure identifier.

We then obtain the following generalized tuples:

$$Z = ia \wedge (a \leq X \leq c) \wedge (b \leq Y \leq d)$$

$$Z = ib \wedge (e \leq X \leq f) \wedge (g \leq Y \leq h)$$

$$Z = ib \wedge (f \leq X \leq l) \wedge (h \leq Y \leq i)$$

where ia and ib are the figure identifiers. This is the only representation allowed by the model considered in [13]. □

In general, constraints are assumed to be in *normal* or *canonical* form [1, 2, 12]. In this paper we do not consider any particular normal form. We suppose that two d-generalized tuples are equivalent if their extensions coincide. Moreover, a generalized relation can contain several equivalent d-generalized tuples only if their representation (eventually, in normal form) is different.

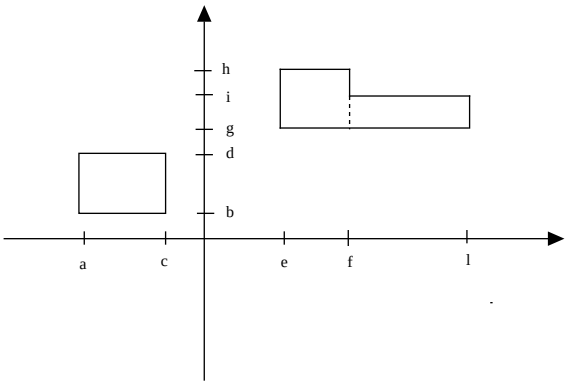

Fig. 1. Figures represented in relation R

A last remark is related to generalized relations dealing with both *constraint variables*, used inside constraints, and *relational variables*, playing the role of relational attributes. Definition 1 introduces the concept of d-generalized tuple by considering that all the variables in a relation schema are constrained. In practice, it can be useful to deal with relations containing both constraint and relational variables. This topic has been addressed in [1], [2] and [19]. By dealing with both constraint and relational variables, we can associate descriptive non-structured knowledge, represented by unconstrained values (i.e., in the case of a geographical system, a river name), with complex data, represented by information modeled by constraints (i.e., the representation by constraints of the river path). Our model can be easily extended to this case (see for example [19]). However, for the sake of simplicity, in the following we consider only generalized databases of the form introduced in Definition 1, without considering descriptive non-structured knowledge.

3 The Generalized Relational Algebra GRA($\Phi, \mathcal{F}$)

Different constraint theories should be used for different application domains (e.g., temporal [15] or spatial [18, 19] applications). In general, the set of operations introduced in the algebra proposed in [12] and in [19] is not adequate for any possible application context. This is due to two main reasons:

- Different application domains may need different specific operators. These operators may not always be modeled by the proposed constraint language.
- If we consider a particular application domain, a logical theory must be chosen with respect to some specific expressiveness and complexity requirements. Often, the chosen theory is not adequate to support all the functionalities needed by the specific application. For example, if we choose linear polynomial constraints to model spatial data, the concept of Euclidean dis-

tance cannot be modeled, even if it can be modeled in the logical theory of (not necessarily linear) polynomial constraints.

In order to consider these aspects in defining a constrained algebra, we introduce a family of languages, parameterized with respect to the chosen logical theory and to the particular application domain.

Each language in this family consists of two sets of operators. The first set is composed of a fixed number of operators, defined regardless of the application domain. The second set includes operators strictly dependent on the chosen domain. Thus, if the generalized relational model is to be used in a spatial context, some operators strictly related to spatial queries are introduced in this second set. However, if we want to model temporal applications, the set of operators to consider may be different.

In order to model this family of languages, the base language we consider is an extension of the constrained relational algebra introduced in [12] and in [19]. We tailor such language to a specific logical theory and to a given application domain. The latter extension is possible by specifying a set of functions $\mathcal{F}$ on d-generalized tuples, assumed to be computable and total (i.e., defined on all the domain values). For example, if we deal with spatial applications and, for efficiency requirements, we choose the theory of linear polynomial constraints [16], we can introduce the concept of Euclidean distance by defining an ad hoc function. Note that the definition of functions in $\mathcal{F}$ may rely on efficient algorithms defined within a specific application domain.

Our approach thus avoids introducing a "complex" logical theory, with high computational complexity. Moreover, it allows to adopt a "simple" logic, for example the linear polynomial inequality constraint theory, and to add the specific functionalities requiring a higher complexity as functions. In this way, the theory still remain simple and the increment of expressibility, and in general of complexity, is embedded in the set of external functions. The expressibility and complexity of the language can be changed by modifying this set. Note that this is different from using a more powerful logical theory, since, in this case, the changes of complexity are not related to the use of a special set of functions. Indeed, they depend on the language as a whole.

In the following, the algebraic operators are introduced. We distinguish between *application independent* operators, defined independently from functions in $\mathcal{F}$, and *application dependent* operators, defined by means of the functions in $\mathcal{F}$. We denote with GRA($\Phi, \mathcal{F}$) (abbreviated as GRA), the family of languages constructed in this way.

3.1 Application Independent Operators

The application independent operators are all those operators that are defined in the languages of the family regardless of the application domain. The obtained application independent language is a generalization of the algebra proposed in [12] and in [19].

Op. name	Op. symbol	Restrictions	Semantics
tuple difference	$r = r_1 \setminus^t r_2$	$\alpha(r_1) = \alpha(r_2) = \alpha(r)$	$r = \{t : \exists t_1 \in r_1,$ $t = t_1 \wedge \neg t_2^1 \wedge ... \wedge \neg t_2^n\}$ $r_2 = \{t_2^1, ..., t_2^n\}$
tuple selection	$r = \sigma_P^t(r_1)$	$\alpha(P) \subseteq \alpha(r_1)$, $\alpha(r) = \alpha(r_1)$	$r = \{t : \exists t_1 \in r_1, t = t_1 \wedge P\}$
set difference	$r = r_1 \setminus^s r_2$	$\alpha(r_1) = \alpha(r_2) = \alpha(r)$	$r = \{t : t \in r_1, \nexists t' \in r_2 :$ $ext(t) = ext(t')\}$
set selection	$r = \sigma^s_{(P,\theta)}(r_1)$	$\alpha(P) \subseteq \alpha(r)$, $\alpha(r) = \alpha(r_1)$	$r = \{t : \exists t \in r_1,$ $ext(t) \theta ext(P)$ is true $\}$
renaming	$r = \varrho_{A\|B}(r_1)$	$A \in \alpha(r), B \notin \alpha(r)$, $\alpha(r) = (\alpha(r_1) \setminus \{A\})$ $\cup \{B\}$	$r = \{t : \exists t' \in r_1,$ $t = t'[A \mid B]\}$
union	$r = r_1 \cup r_2$	$\alpha(r_1) = \alpha(r_2) = \alpha(r)$	$r = \{t : t \in r_1$ or $t \in r_2\}$
projection	$r = \Pi_{[x_{i_1},...,x_{i_p}]}(r_1)$	$\alpha(r_1) = \{x_1, ..., x_m\}$, $1 \leq i_1, ..., i_p \leq m$, $\alpha(\Pi_{[x_{i_1},...,x_{i_p}]}(r_1)) =$ $\{x_{i_1}, ..., x_{i_p}\}$	$r = \{\pi_{[x_{i_1},...,x_{i_p}]}(t) : t \in r_1\}$ $\pi_{[x_{i_1},...,x_{i_p}]}(t) = \exists x_1, ..., \exists x_m$ $(t \wedge \bigwedge_{l=1}^{p} y_l = x_{i_l})$
natural join	$r = r_1 \bowtie r_2$	$\alpha(r) = \alpha(r_1) \cup \alpha(r_2)$	$r = \{t : \exists t_1 \in r_1, \exists t_2 \in r_2,$ $t = t_1 \wedge t_2\}$

Table 1. Application independent operators

The introduced generalizations are related to a different way of *seeing* d-generalized tuples. Indeed, d-generalized tuples can be considered under two different points of view: as a finite representation of an infinite set of relational tuples, or as a single object, having a meaning by itself.

For example, consider a generalized relation $R(X, Y)$ where each generalized tuple represents a rectangle. Each tuple will have the form: $X \geq a_1 \wedge X \leq a_2 \wedge Y \geq b_1 \wedge Y \leq b_2$. If we want to determine the set of points contained in the intersection space of two given rectangles, each constraint should be considered as the finite representation of an infinite set of points. However, if we want to know which rectangles are contained in a given space, each constraint must be considered as a single object. In the latter case, all the points of a single rectangle must satisfy a certain condition.

By taking these two different points of view into account, we define two sets of operators: *tuple operators* and *set operators*. Set operators consider d-generalized tuples as single objects and assume a sort of universal quantification over all the relational tuples representing a single d-generalized tuple. Set operators avoid the use of explicit tuple identifiers. From a user point of view, they simplify the usage of d-generalized tuples.

Table 1 presents application independent operators[4]. Following the approach

[4] Note that some conditions in the table use function ext. An equivalent logical representation of these conditions is however possible. For example, the condition

proposed in [11], each operator is described by using two kinds of clauses: those presenting the *schema restrictions* required by the argument relations and by the result relation, and those introducing the operator semantics. In the table, α is a function that takes a relation or a d-generalized tuple and returns their schema, i.e. the set of variables appearing in it.

The operators are the following:

1. *Tuple difference*: Given two generalized relations r_1 and r_2, the operator returns all the relational tuples in r_1 not contained in r_2. The condition requires that if c is a constraint of a certain logical theory, then $\neg c$ is equivalent to a certain constraint c', definable in the same theory. For example, given a theory with the predicate $<$, we need the theory to contain the predicate $\geq$. For a more formal definition of tuple difference, see [12].
2. *Tuple selection*: Let P be a d-generalized tuple on the considered logical theory. This operator returns all the relational tuples, represented in the generalized relation, that satisfy P, i.e., belonging to the extension of P.
3. *Set difference*: Given two generalized relations r_1 and r_2, this operator returns all the d-generalized tuples contained in r_1 for which there does not exists an equivalent d-generalized tuple contained in r_2.
4. *Set selection*: This operator selects all the d-generalized tuples satisfying a certain condition from a generalized relation. The condition has the form (P, θ), where P is a d-generalized tuple on the considered logical theory and $\theta \in \{\subseteq, \supseteq, (\bowtie \neq \emptyset), (\bowtie = \emptyset)\}$.
   The set selection operator with condition (P, θ), applied on a generalized relation r, selects from r only the d-generalized tuples for which there exists a relation θ between the extension of the d-generalized tuple and the extension of P. The possible meanings of θ operators are:
   - $\theta = \subseteq$: we select all the d-generalized tuples in r whose extension is contained in the extension of P (i.e., those for which, if $\alpha(t) = \tilde{x}$, $\forall \tilde{x}(t \rightarrow P)$ holds);
   - $\theta = \supseteq$: we select all the d-generalized tuples in r whose extension contains the extension of P (i.e., those for which, if $\alpha(t) = \tilde{x}$, $\forall \tilde{x}\ (P \rightarrow t)$ holds);
   - $\theta = (\bowtie = \emptyset)$ $[\theta = (\bowtie \neq \emptyset)]$: we select all the d-generalized tuples in r whose natural join with P is [is not] empty (i.e., those for which $t \wedge P$ is inconsistent [$t \wedge P$ is consistent]). Thus, we select all the d-generalized tuples whose extension has no [at least one] relational tuple in common with the extension of P.
5. *Renaming*: This operation changes some of the variable names of a generalized relation. In the table, $t[A \mid B]$ identifies the d-generalized tuple obtained from t by renaming variable A with B.
6. *Union*: Given two generalized relations r_1 and r_2, it generates the union of the input generalized relations.
7. *Projection*: Given a generalized relation r, this operator returns all the d-generalized tuples restricted to a subset of the variables of r. In the table, t

---

$ext(t) \subseteq ext(P)$ can be rewritten as $\forall x\ (t \rightarrow P)$.

is a d-generalized tuple on variables $x_1, ..., x_m$, $1 \leq i_1, ..., i_p \leq m$; $y_1, ..., y_p$ are variables different from $x_{i_1}, ..., x_{i_p}$ and $\pi_{[x_{i_1},...,x_{i_p}]}(t) = \exists x_1, ..., \exists x_m(t \wedge \bigwedge_{l=1}^{p} y_l = x_{i_l})$. In the table, we use the symbol Π to denote the projection operation on a generalized relation, and π to denote projection on a single d-generalized tuple.

8. *Natural Join*: This operation represents the generalization of the relational natural join for the generalized model. Given two generalized relations r_1 and r_2, for any d-generalized tuple $t_1 \in r_1$ and for any d-generalized tuple $t_2 \in r_2$, this operator returns the natural join of all the relational tuples represented by t_1 with those represented by t_2.
   Note that we have defined the natural join operator also on disjoint schemas. As we will see, this assumption simplifies the definition of the Cartesian product (see Subsection 3.3).

For the renaming operator, no distinction between set and tuple semantics exists. For union, projection and natural join, we propose only one semantics, corresponding to the most intuitive one. However, tuple and set versions of these operators could be given, by assuming to remove duplicates. Indeed, the concept of duplicate from a tuple point of view may be different from the concept of duplicate from a set point of view. In this paper, we assume to maintain duplicates.

Example 2. Let r_1 and r_2 be two relations on the schema X, Y. Each relation contains generalized tuples defining rectangles. Figure 2 shows relation r_1 (white) and relation r_2 (shadow). Then:

1. $r_1 \setminus^s r_2$ returns all generalized tuples, that is, all the *rectangles*, contained in r_1 and not in r_2. In this case, it returns all the generalized tuples of r_1, since no tuple of r_1 is also a tuple of r_2.
2. $r_1 \setminus^t r_2$ returns all *points* (corresponding to relational tuples) contained in r_1 and not in r_2. Thus, it returns all the (parts of) rectangles contained in r_1 that are not contained in r_2 (see Figure 3).
3. $\sigma^s_{X \geq 3}(r_1)$ returns all *rectangles* contained in r_1 with $X \geq 3$. In this case, it returns only one rectangle (see Figure 4).
4. $\sigma^t_{X \geq 3}(r_1)$ returns all *points* contained in r_1 such that $X \geq 3$ (see Figure 5). □

3.2 Application Dependent Operators

Let Φ be a decidable logical theory. Let $\mathcal{F}$ be a set of total computable functions such that $\forall f \in \mathcal{F}$, $f : A \rightarrow B$, $A, B \subseteq DOM_{gentuple}(\Phi)$. $DOM_{gentuple}(\Phi)$ is the set of all the possible d-generalized tuples defined on the logical constraint theory Φ. Functions in $\mathcal{F}$ can be considered as library functions, completing the knowledge of a certain application domain. We assume that, if $\mathcal{F} \neq \emptyset$, then it contains at least the projection operator. Application dependent operators for the language GRA$(\Phi, \mathcal{F})$ are summarized in Table 2:

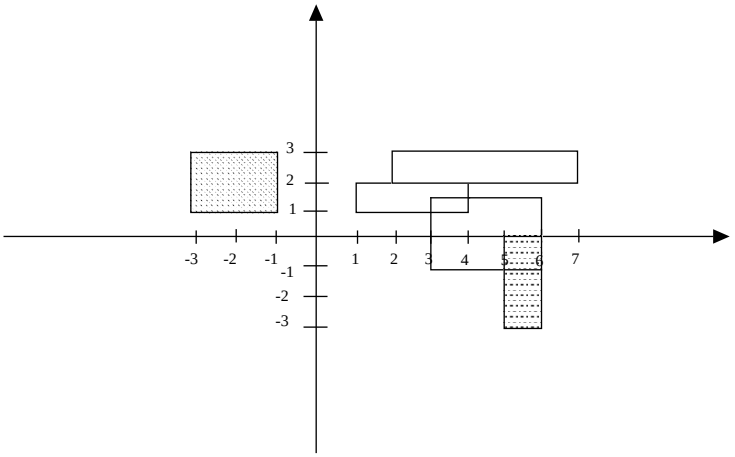

Fig. 2. Relation r_1 (white) and r_2 (shadow)

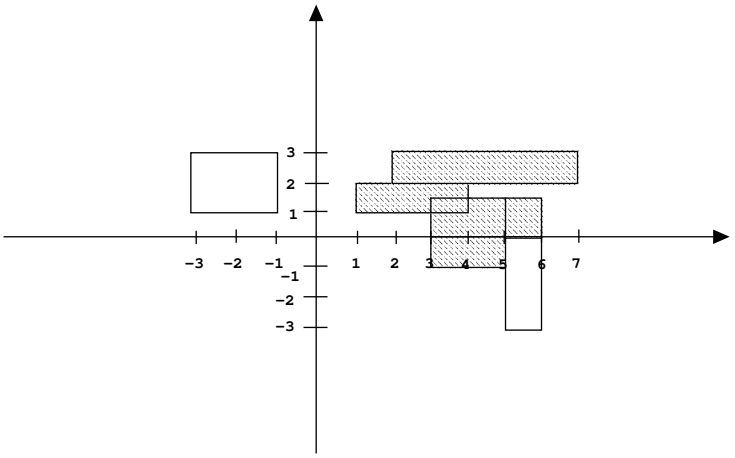

Fig. 3. Relation $r_1 \setminus^t r_2$ (shadow)

1. *Apply Transformation*: For each $f \in \mathcal{F}$, we introduce an operator AT_f (AT as Apply Transformation). The operator takes a relation and returns another relation by replacing each d-generalized tuple t by the d-generalized tuple $f(t)$.
2. *Application dependent set selection*: A set selection operator using functions in $\mathcal{F}$ can be defined. To this purpose, we introduce the condition (f_1, f_2, θ), where $f_1, f_2 \in \mathcal{F}$ and $\theta \in \{\subseteq, \supseteq, (\bowtie \neq \emptyset), (\bowtie = \emptyset)\}$.
   The operator selects only the d-generalized tuples for which there exists a relation θ between the extension of the d-generalized tuple, transformed by f_1, and the extension of the d-generalized tuple, transformed by f_2.

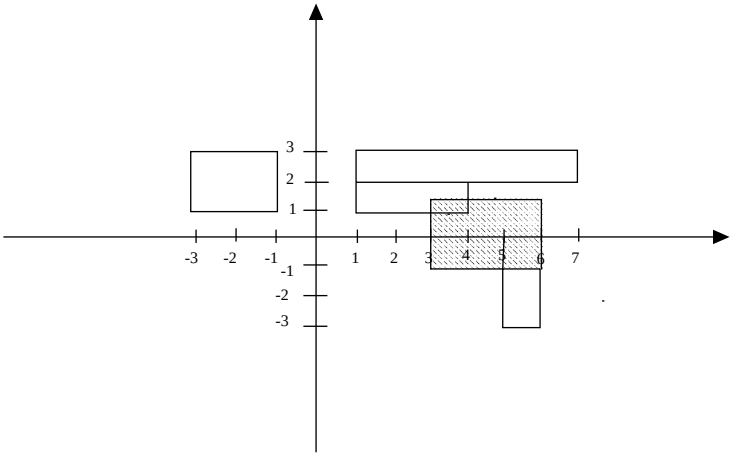

Fig. 4. Relation $\sigma^s_{X\geq 3}(r_1)$ (shadow)

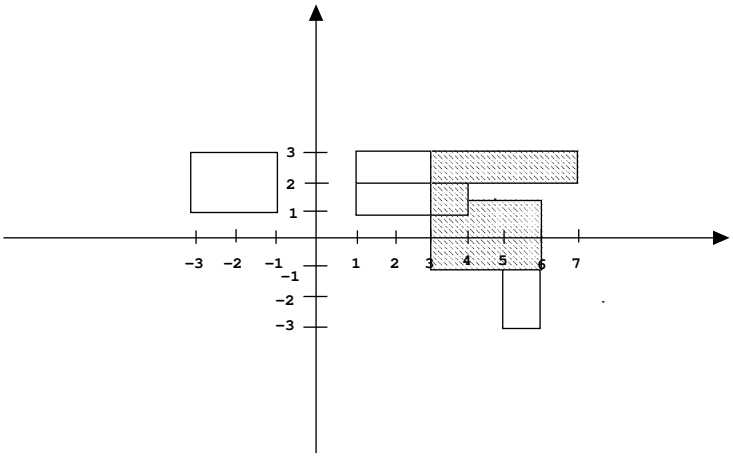

Fig. 5. Relation $\sigma^t_{X\geq 3}(r_1)$ (shadow)

Note that the application dependent set selection differs from the (application independent) set selection only for the fact that conditions depending on the application domain (i.e., depending on $\mathcal{F}$) are used in this case.

3.3 Derived Operators

By using the previous operations, we can define some useful derived operations, whose semantics is described in Table 3. In particular:

Op. name	Op. symbol	Restrictions	Semantics
apply transformation	$r = AT_f(r_1)$		$r = \{t : f \in \mathcal{F}, \exists t' \in r_1 : t = f(t')\}$
set selection	$r = \sigma^s_{(f_1,f_2,\theta)}(r_1)$	$\alpha(r) = \alpha(r_1)$	$r = \{t : \exists t \in r_1,$ $ext(f_1(t))\theta ext(f_2(t))$ is true $\}$

Table 2. Application dependent operators

Op. name	Op. symbol	Restrictions	Derived sem.
Cartesian product	$r = r_1 \times r_2$	$\alpha(r) = \alpha(r_1) \cup \alpha(r_2)$, $\alpha(r_1) \cap \alpha(r_2) = \emptyset$	$r = r_1 \bowtie \varrho_{r_1}(r_2)$
tuple intersection	$r = r_1 \cap^t r_2$	$\alpha(r) = \alpha(r_1) = \alpha(r_2)$	$r = r_1 \bowtie r_2$
set intersection	$r = r_1 \cap^s r_2$	$\alpha(r) = \alpha(r_1) = \alpha(r_2)$	$r = r_1 \setminus^s (r_1 \setminus^s r_2)$
derived selection	$r = \sigma^s{}_{(=\emptyset)}(r_1)$	$\alpha(r) = \alpha(r_1)$	$r = \sigma^s_{(inc,\subseteq)}(r_1)$
	$r = \sigma^s{}_{(\neq\emptyset)}(r_1)$	$\alpha(r) = \alpha(r_1)$	$r = r_1 \setminus^s (\sigma^s{}_{(=\emptyset)}(r_1))$

Table 3. Derived operators

- *Cartesian product*: The Cartesian product can be obtained from the natural join, if the two considered relations have disjoint schemas[5]. If they do not have disjoint schema, we rename the variables of the second relation. In Table 3, the operation $\varrho_{r_1}(r_2)$ renames all the variables of the schema of r_2 that are also contained in the schema of r_1.
- *Tuple intersection*: Given two generalized relations r_1 and r_2, this operation returns all the relational tuples contained in the extension of both relations.
- *Set intersection*: Given two generalized relations r_1 and r_2, this operation returns the d-generalized tuples contained in both relations.
- *Derived selection*: Two other conditions can be inserted in the definition of the set selection operators. They are: $(\neq \emptyset)$ and $(= \emptyset)$. They can be obtained as follows:
  - $\sigma^s{}_{(=\emptyset)}(r) = \sigma^s_{(inc,\subseteq)}(r)$, where *inc* is an inconsistent constraint on the considered logical theory. This operation selects all the inconsistent tuples contained in r.
  - $\sigma^s{}_{(\neq\emptyset)}(r) = r \setminus^s (\sigma^s{}_{(=\emptyset)}(r))$. This operation selects all the consistent tuples contained in r.

Let D be a generalized database schema. Let Φ be a decidable logical theory. Let $\mathcal{F}$ be a set of total recursive functions. The GRA$(\Phi, \mathcal{F})$ language for D is composed of all the expressions definable by using the operators introduced in Table 1, 2 and 3, applied to all the relations in D as well as to relations

[5] This definition is possible because our definition of natural join still works on disjoint schemas.

implicitly defined by constraints. Functions contained in $\mathcal{F}$ can also be applied to constraints explicitly used in GRA expressions. For example, if $f \in \mathcal{F}$, the expression $R(X,Y) \wedge f(Z = X + Y)$ is a GRA$(\Phi, \mathcal{F})$ expression for the database schema containing a generalized relation R over X and Y.

3.4 Set and Tuples Operators: Examples and Remarks

Set operators are of particular interest when it is necessary to treat sets of relational tuples (implicitly represented by generalized tuples), as single objects with a well-defined identity. This feature has proved to be useful in spatial databases, where relational tuples with k variables represent points in a k-dimensional space and d-generalized tuples represent objects (i.e. point-sets) embedded in this space. In such context, set-oriented operations improve significantly the usability of the language and reduce the complexity of the expressions when writing the most common queries.

However, *set* operators do not add expressive power to the language. Let GRA$_{tuple}(\Phi)$ denote the generalized relational algebra obtained from GRA$(\Phi, \mathcal{F})$ by considering an empty set of functions and by removing set operators. For linear polynomial constraint theory, this algebra is equivalent to the one proposed in [12]. Let $\mathcal{C}$ be the generalized relational calculus proposed in [13]. The following examples show how queries involving set operators can be translated in expressions of the calculus $\mathcal{C}$ and in the algebra GRA$_{tuple}(\Phi)$.

Both examples refer to relations R and S on the schema X, Y. In order to simulate set operators in $\mathcal{C}$ or GRA$_{tuple}(\Phi)$, the schema of relations R and S should be modified. In particular, since it is necessary to identify distinct sets of relational tuple, represented by distinct generalized tuples, as distinct objects, an additional variable has to be introduced. Thus, R and S are mapped to relations R' and S' on the schema N, X, Y. N is a generalized tuple identifier. As the models proposed in [12] and in [13] do not deal with d-generalized tuples, we assume that relations R and S contain non-disjunctive generalized tuples. Moreover, we assume that inconsistent tuples are directly removed by the computation.

Example 3. Consider the query *"Find all objects in R that are contained in the object o"* (see Section 5). Let P be the generalized tuple representing o in the Euclidean space. This query is expressed in $GRA(\Phi, \mathcal{F})$ using a set operator, as follows:

$$\sigma^{s}_{(P,\subseteq)}(R)$$

in $\mathcal{C}$ as follows:

$$(\forall \tilde{x}, \tilde{y}\ R'(n, \tilde{x}, \tilde{y}) \rightarrow P) \wedge R'(n, x, y)$$

in GRA$_{tuple}(\Phi)$ as follows:

$$R' \setminus^{t} (R' \bowtie (\Pi_{[N]}(R' \setminus^{t} \sigma^{t}_{P}(R')))).$$

Previous expression has the following meaning:

- $\sigma^{t}_{P}(R')$ selects the points contained in P, together with the identifier of the object to which they belong.

- $R' \setminus^t \sigma^t_P(R')$ selects the points that are not contained in P, together with the identifier of the object to which they belong. These objects are not contained in P.
- $\Pi_{[N]}(R' \setminus^t \sigma^t_P(R'))$ selects the identifiers of the objects not contained in P.
- $R' \bowtie (\Pi_{[N]}(R' \setminus^t \sigma^t_P(R')))$ selects the identifier and the extension of the objects not contained in P.
- $R' \setminus^t (R' \bowtie (\Pi_{[N]}(R' \setminus^t \sigma^t_P(R'))))$ selects the objects contained in P. □

Example 4. Consider the query *"Find all pairs of object belonging to R and S, such that their intersection is not empty"* (Spatial Join) (see Section 5). Assuming the Cartesian product renames variables X, Y in variables X', Y', this query is expressed in $GRA(\Phi, \mathcal{F})$ using a set operator, as follows[6]:

$$\sigma^s_{(\pi_{[X,Y]}, \varrho_{[X'|_X, Y'|_Y]} \circ \pi_{[X',Y']}, (\bowtie \neq \emptyset))}(R \times S)$$

in $\mathcal{C}$ as follows[7]:

$$(\exists \tilde{x}, \tilde{y} : R'(n_1, \tilde{x}, \tilde{y}) \wedge S'(n_2, \tilde{x}, \tilde{y})) \wedge R'(n_1, x, y) \wedge S'(n_2, x', y')$$

in $\mathrm{GRA}_{tuple}(\Phi)$ as follows:

$$R' \bowtie (\Pi_{[N,N']}(R' \bowtie \varrho_{[N|_{N'}]}(S'))) \bowtie \varrho_{[N|_{N'}, X|_{X'}, Y|_{Y'}]}(S').$$

Previous expression has the following meaning:

- $R' \bowtie \varrho_{[N|_{N'}]}(S')$ selects generalized tuples over variables N, X, Y, N', X, Y, representing the points belonging both to an object of relation R, identified by N, and to an object of relation S, identified by N'.
- $\Pi_{[N,N']}(R' \bowtie \varrho_{[N|_{N'}]}(S'))$ selects the identifiers of pairs of intersecting objects.
- $R' \bowtie (\Pi_{[N,N']}(R' \bowtie \varrho_{[N|_{N'}]}(S')))$ selects generalized tuples over variables N, X, Y, N', representing the objects contained in relation R, identified by N, X and Y, together with the identifier N' of an object contained in relation S, having a non-empty intersection with it.
- $R' \bowtie (\Pi_{[N,N']}(R' \bowtie \varrho_{[N|_{N'}]}(S'))) \bowtie \varrho_{[N|_{N'}, X|_{X'}, Y|_{Y'}]}(S')$ selects all pairs of intersecting objects, belonging respectively to R and S. □

As we can see from both examples, set operators support more compact forms to express queries. Moreover, the used operators seems to be more suitable from a user point of view.

---

[6] Note that the used operator $\varrho_{[X'|_X, Y'|_Y]} \circ \pi_{[X',Y']}$ can be defined as a function contained in $\mathcal{F}$. In this case, the user does not have to deal with renaming.

[7] Operator $\varrho_{[N|_{N'}, X|_{X'}, Y|_{Y'}]}$ is equivalent to $\varrho_{[N|_{N'}]}(\varrho_{[X|_{X'}]}(\varrho_{[Y|_{Y'}]}))$.

4 Spatial Data in Constraint Databases

Spatial databases involve complex data which are composed of geometry, embedded in spaces of various dimensions, and alphanumeric information, which are linked to geometry at various levels of granularity.

The coexistence of geometry and alphanumeric data implies the following basic requirements for any data model which has to describe spatial data:

R1. It should be able to represent the association between alphanumeric information and geometry at any level of granularity, that is, considering both simple geometric objects, such as points, segments or convex polygons, and also complex geometries, obtained, for example, through an aggregation of simple geometric objects.
R2. It should be able to integrate in a unified structure the geometric and alphanumeric aspects of spatial data [17].

In this section, we analyze the issue concerning the modeling of spatial data in the generalized relational model. In particular, we want to exploit the constraint formalism in order to fulfill the requirements discussed above. Indeed:

- constraints finitely represent infinite sets of points embedded in n-dimensional spaces, thus a single constraint can model complex geometries;
- combination of constraints can represent in a simple way bindings between spatial data and descriptive data at different levels of granularity (requirement R1);
- no "ad-hoc" concept has to be added to the constraint formalism in order to model spatial data (requirement R2).

4.1 Representing Geometry Using Constraints

In the proposed generalized relational model, a relation contains a set of constraints over a finite set of variables $X_1, ..., X_n$, which represents the schema of the relation itself. In order to represent geometry in this model, the idea is to use a relation with n variables to contain points of an n-dimensional space. In this way, d-generalized tuples of this relation represent sets of points (i.e. spatial objects) embedded in that space.

The different expressive power of different classical logical theories permits to represent several types of spatial objects within a large range of complexity. In this paper, we consider the following constraint theory, for the representation of spatial data.

Definition 2. *(Linear Polynomial Inequality Constraint Theory Φ_P)*: The theory Φ_P represents formulas of the form $p(X_1, ..., X_n)\ \theta\ 0$, where p is a linear polynomial with real coefficients in variables $X_1, ..., X_n$ and $\theta \in \{=, \neq, \leq, <, \geq, >\}$. ◇

Note that the definition of d-generalized tuple has some consequences in representing spatial data. Indeed, a single generalized tuple (i.e., conjunctions of constraints in Φ_P) can represent only *convex geometric objects*. This means, for example, that a line composed of several concatenated segments cannot be represented as one generalized tuple, but only as a set of generalized tuples.

In order to overcome such limitation, we have defined (see Definition 1) d-generalized tuples as a disjunction of conjunctions of constraints, thus obtaining more expressive power at the level of a single generalized tuple. For example, we can represent concave geometric objects with a single d-generalized tuple.

An alternative way to overcome this problem is to represent sets of spatial objects embedded in an n-dimensional space through a relation which contains $n+1$ variables. The first n variables represent points of the space, the $(n+1)$-th variable is used as an identifier for the object. In this case, complex objects can be represented by several generalized tuples, where tuples belonging to the same object have the same identifier. However, this solution has some drawbacks: (a) it enforces the decomposition of spatial data; (b) it requires the management of the identifiers; (c) it is more expensive in terms of space.

Another issue concerning the management of spatial data in a database is the representation of topological relationships among objects [6, 7]. Topological relationships are the basis of a large number of spatial queries and their explicit representation can be useful for query optimization. A particular "hierarchical" structure where complex spatial objects (e.g. higher-dimensional entities) are defined in terms of simpler objects (e.g. in terms of lower-dimensional ones), is a possible approach to represent topology. In this way, the implementation of some topological properties, such as adjacency, can be simplified by testing whether two entities share some lower-dimensional entity. However, the mapping of such hierarchical structure in a constraint database requires the introduction of some integrity constraints between d-generalized tuples representing objects of different levels. Thus, two different spatial data representations in a constraint database are possible:

E1. a representation of spatial data in a "flat" structure, that is, without any integrity constraints among relations;
E2. a representation of spatial data in a "hierarchical" structure that requires integrity constraints.

Since the E2 representation can be modeled as a particular case of E1, we discuss E1 into more details.

4.2 Spatial Data Types in E^2: a Mapping to d-Generalized Tuples

In order to illustrate an example of representation of spatial data in a constraint database, we define three spatial data types which are usually the basis of any data model for spatial objects embedded in the Euclidean plane E^2. Note that we restrict our consideration to two-dimensional examples, for simplicity. However, the same definitions could be easily extended to higher dimensions.

We define each data type by giving the representation of its values in vector form together with the mapping *map* of these values to d-generalized tuples:

$$map:\ DOM_{vector} \rightarrow DOM^2_{gentuple}$$

where: DOM_{vector} = SPT $\cup$ LN $\cup$ PG; SPT, LN, PG are defined below, and $DOM^2_{gentuple}$ is the domain of the d-generalized tuples on Φ_P with two variables.

Definition 3. *(SET POINT TYPE (SPT))*: Its values are *finite* sets of points in the Euclidean plane E^2:

$$\begin{aligned} SPT &= \{P : P \subset E^2 \wedge card(P) < \infty\} \\ map(P) &= map(p_1) \vee ... \vee map(p_m) \\ &\quad \text{where } P \in SPT,\ p_i \in P \text{ and } m = card(P) \\ map(\{< x_i, y_i >\}) &= (X = x_i \wedge Y = y_i) \quad \text{where } < x_i, y_i >= p_i. \end{aligned}$$

$card(P)$ denotes the cardinality of the set P. ◇

Definition 4. *(LINE TYPE (LN))*: Its values are chains (i.e. sequences) of directed segments of the Euclidean plane E^2 such that non-consecutive segments cannot intersect. Besides, the first endpoint of each segment s coincides with the second endpoint of the segment preceding s in the chain and the second endpoint of s coincides with the first endpoint of the segment following s in the chain.

$$\begin{aligned} LN = \{< P_1, ..., P_n >_{LN}:\ &(P_i \in E^2,\ 1 \leq i \leq n) \wedge \\ &(\overline{P_i P_{i+1}} \cap \overline{P_j P_{j+1}} = \emptyset,\ i \neq j,\ 1 \leq i, j \leq n-1)\} \end{aligned}$$

where $\overline{P_i P_{i+1}}$ is the set of points representing the segment joining P_i to P_{i+1} without endpoints.

$$\begin{aligned} map(< P_1, ..., P_n >_{LN}) = &((x_1 \leq X \wedge X \leq x_2 \wedge a_1 X + b_1 Y + c_1 = 0) \vee ... \\ &\vee (x_{n-1} \leq X \wedge X \leq x_n \wedge a_{n-1} X + b_{n-1} Y + c_{n-1} = 0)) \end{aligned}$$

where $< P_1, ..., P_n >_{LN} \in LN$, $P_i =< x_i, y_i >$ and a_j, b_j, c_j are the real coefficients of the equation $(a_j X + b_j Y + c_j = 0)$ representing the straight line passing through P_j and P_{j+1}. ◇

Definition 5. *(POLYGON TYPE (PG))*: Its values are simple polygons. Simple polygons define polygonal regions. A polygonal region is a closed connected and regular subset of the Euclidean plane E^2 [20]. A simple polygon can be defined by a closed chain of segments. We suppose by convention that the polygonal region is always on the left side of each directed segment of the polygon bounding it.

$$\begin{aligned} PG = \{< P_1, ..., P_n >_{PG}:\ &(P_i \in E^2,\ 1 \leq i \leq n)\ \wedge (P_1 = P_n)\ \wedge \\ &(\overline{P_i P_{i+1}} \cap \overline{P_j P_{j+1}} = \emptyset,\ i \neq j,\ 1 \leq i, j \leq n-1)\} \end{aligned}$$

We consider a particular mapping which assumes to have a partition of the polygon into convex polygons[8]. More formally:

$$map(pg) \;=\; map(c_1) \;\vee ... \vee\; map(c_n)$$

where: $pg \in PG$ and $c_1, ..., c_n$ represents the partition of pg into convex polygons.

$$map(pg_{convex}) = ((a_1X + b_1Y + c_1)sign(\overline{P_1P_2}) \geq 0) \wedge ... \\ \wedge((a_nX + b_nY + c_n)sign(\overline{P_nP_1}) \geq 0)$$

where: $pg_{convex} =< P_1, ..., P_n >$, $P_i =< x_i, y_i >$, a_j, b_j, c_j are the real coefficients of the equation $(a_jX + b_jY + c_j = 0)$ representing the straight line passing through P_j and P_{j+1} and $sign(\overline{P_hP_k})$, where either $h = i, k = i + 1$, for $1 \leq i \leq n - 1$, or $h = n, k = 1$, is defined as follows:

$$sign(\overline{P_hP_k}) \;=\; \begin{cases} 1 \;\; (x_k - x_h > 0) \vee (x_k - x_h = 0 \wedge y_k - y_h < 0) \\ -1 \; (x_k - x_h < 0) \vee (x_k - x_h = 0 \wedge y_k - y_h > 0). \end{cases} \diamond$$

The introduction of the function $sign()$ is necessary in order to take into account that the polygonal region represented by a simple polygon is always on the left side of the polygon itself.

4.3 Spatial Transformations

When GRA is applied to a specific application area, it is necessary to specify the logical theory Φ, which defines the constraints, and the set of transformation operators $\mathcal{F}$, to be used as parameters of application dependent operators of the generalized relational algebra.

In the spatial application area, the need of introducing these operators derives from the analysis of the most common queries and processing tasks. Since application independent operators applied on d-generalized tuples cannot model all the needed functions, new spatial transformations have to be added to the base language.

Definition 6. *(Spatial transformations)*[9]: Given a spatial object sp belonging to DOM_{vector}, the following spatial transformations are defined:

- $Boundary : DOM_{vector} \rightarrow DOM_{vector}$
  $Boundary(sp)$ returns the spatial object that represents the "limit points" of sp. There are different definitions of boundary. The definition we adopt here is obtained by giving emphasis to the dimensionality of the spatial object.

[8] Note that such partition is always possible, for example by using a Delaunay triangulation [20].

[9] Note that functions $IsLinear$ and $IsPolygon$ can be simply obtained from function $Boundary$. We permit this redundancy as the set of functions we propose has been designed with no minimality and completeness requirements in mind.

This implies, for example, that the boundary of a segment is composed of its endpoints. More formally:

$$Boundary(sp) = \begin{cases} \emptyset & \text{if } sp \in SPT \\ (\text{endpoints of } sp) & \text{if } sp \in LN \\ (\text{border of } sp) & \text{if } sp \in PG \end{cases}$$

Giving emphasis to the dimensionality D of the embedding space (for E^2, $D = 2$), the boundary of a segment is the segment itself. This is also the main difference between this definition and the one reported in [18], where the boundary of a spatial object is defined by using a constraint in a non-linear logical theory.

- $Interior : DOM_{vector} \rightarrow DOM_{vector}$
  $Interior(sp)$ returns the set of points of sp which are not boundary points. Thus, it returns sp, if $\{sp\} \in SPT$; the line without its endpoints, if $sp \in LN$; the polygonal region without its border, if $sp \in PG$.
- $Buffer : DOM_{vector} \times \Re \rightarrow DOM_{vector}$
  $Buffer(sp, r)$ returns the polygon $ps \in PG$ that represents the set of points which have a distance less than r from the boundary of sp. For uniformity, we assume to have a function
  $Buffer_r : DOM_{vector} \rightarrow DOM_{vector}$ for each $r \in \Re$. $\Re$ represents real numbers.
- $IsLinear : DOM_{vector} \rightarrow DOM_{vector}$
  $IsLinear(sp)$ returns sp if sp is a line, an empty set otherwise.
- $IsPolygon : DOM_{vector} \rightarrow DOM_{vector}$
  $IsPolygon(sp)$ returns sp if sp is a polygon, an empty set otherwise.
- $Area : DOM_{vector} \rightarrow DOM_{vector} \times \Re$
  $Area(sp)$ returns sp and its corresponding area. ◇

Since the representation of a spatial object in constraint databases is a d-generalized tuple, all spatial transformations must be defined in this model as transformations on d-generalized tuples.

Definition 7. Let $DOM^n_{gentuple}$ be the subset of $DOM_{gentuple}(\Phi_P)$ containing exactly all the d-generalized tuples on Φ_P with n variables. For the spatial transformations defined in Definition 6, the following transformation operators on d-generalized tuples are defined (in the following, map^{-1} denotes the inverse of function map):

- Bnd for $Boundary$, $Bnd : DOM^2_{gentuple} \rightarrow DOM^2_{gentuple}$
  $\alpha(t) = \alpha(Bnd(t)) \quad map^{-1}(Bnd(t)) = Boundary(map^{-1}(t)))$.
- Int for $Interior$, $Int : DOM^2_{gentuple} \rightarrow DOM^2_{gentuple}$
  $\alpha(t) = \alpha(Int(t)) \quad map^{-1}(Int(t)) = Interior(map^{-1}(t)))$.
- Buf_r for $Buffer_r$, $Buf_r : DOM^2_{gentuple} \rightarrow DOM^2_{gentuple}$
  $\alpha(t) = \alpha(Buf_r(t)) \quad map^{-1}(Buf_r(t)) = Buffer_r(map^{-1}(t)))$.
- $IsLin$ for $IsLinear$, $IsLin : DOM^2_{gentuple} \rightarrow DOM^2_{gentuple}$
  $\alpha(t) = \alpha(IsLin(t)) \quad map^{-1}(IsLin(t)) = IsLinear(map^{-1}(t)))$.

- $IsPoly$ for $IsPolygon$, $IsPoly : DOM^2_{gentuple} \rightarrow DOM^2_{gentuple}$
  $\alpha(t) = \alpha(IsPoly(t))$ $\quad map^{-1}(IsPoly(t)) = IsPolygon(map^{-1}(t)))$.
- Ar for $Area$, $Ar : DOM^2_{gentuple} \rightarrow DOM^3_{gentuple}$
  $\alpha(Ar(t)) = \alpha(t) \cup \{A\}$ $\quad Ar(t) = t \wedge A = Area(map^{-1}(t))$. $\diamond$

As an example, we report hereby the formal definition of the Bnd function.

Definition 8. *(Transformation operator Bnd for Boundary computation)*: Given a d-generalized tuple $t = c_1 \vee ... \vee c_n$ $(\alpha(t) = \{X, Y\})$, the corresponding d-generalized tuple which contains the boundary of the spatial object is obtained by applying the transformation function defined as follows.
We assume that generalize tuples, and therefore t, are represented in one of the following forms:

$$
\begin{aligned}
c_{point} &= (X = x_0 \wedge Y = y_0) \\
c_{segment} &= (x_1 \leq X \wedge X \leq x_2 \wedge aX + bY + c = 0) \\
c_{convex} &= (a_1X + b_1Y + c_1 \geq 0 \wedge ... \wedge a_nX + b_nY + c_n \geq 0)
\end{aligned}
$$

The definition of the function is given incrementally[10]:

$$
\begin{aligned}
Bnd(\emptyset) &= inc \\
Bnd(c_{point}) &= inc \\
Bnd(c_{segment}) &= (X = x_1 \wedge y = \frac{c + ax_1}{-b}) \vee (X = x_2 \wedge y = \frac{c + ax_2}{-b}) \\
Bnd(c_{convex}) &= (x_{1,1} \leq X \wedge X \leq x_{1,2} \wedge a_1X + b_1Y + c_1 = 0) \vee \\
&\quad \vee (x_{n,1} \leq X \wedge X \leq x_{n,2} \wedge a_nX + b_nY + c_n = 0) \\
&\quad (x_{i,1}, y_1), (x_{i,2}, y_2) \in ext(c_{convex}) \wedge \\
&\quad (\exists j_1, j_2 \in [1..n] : i \neq j_1, j_2 \wedge \\
&\quad (x_{i,1}, y_1) = \begin{cases} a_iX + b_iY + c_i = 0 \\ a_{j_1}X + b_{j_1}Y + c_{j_1} = 0 \end{cases} \\
&\quad (x_{i,2}, y_2) = \begin{cases} a_iX + b_iY + c_i = 0 \\ a_{j_2}X + b_{j_2}Y + c_{j_2} = 0) \end{cases}
\end{aligned}
$$

$$
\begin{aligned}
Bnd(c_{segment_1} \vee ... \vee c_{segment_n}) &= NoDuplicates_{PT}(Bnd(c_{segment_1}) \vee ... \quad (1) \\
&\quad ... \vee Bnd(c_{segment_n})) \\
Bnd(c_{convex_1} \vee ... \vee c_{convex_n}) &= NoDuplicates_{LN}(Bnd(c_{convex_1})) \vee ... \quad (2) \\
&\quad ... \vee Bnd(c_{convex_n}))
\end{aligned}
$$

[10] In the following, inc represents an inconsistent constraint.

$$Bnd(c_1 \vee ... \vee c_n) = Bnd(c_{i_1}) \vee ... \vee Bnd(c_{i_j}) \vee Bnd(c_{i_{j+1}} \vee ... \vee c_{i_k}) \vee \quad (3)$$
$$Bnd(c_{i_{k+1}} \vee ... \vee c_{i_n})$$
$$c_{i_1}...c_{i_j} : \text{ of type } \mathrm{c}_{\mathrm{point}}$$
$$c_{i_{j+1}}...c_{i_k} : \text{ of type } \mathrm{c}_{\mathrm{segment}}$$
$$c_{i_{k+1}}...c_{i_n} : \text{ of type } \mathrm{c}_{\mathrm{convex}}$$

where:

- $NoDuplicates_{PT}$: given a disjunction of conjunctions, where each conjunction represents two points, it eliminates the duplicated conjunctions, leaving only the conjunctions that appear once.
- $NoDuplicates_{LN}$: given a disjunction of conjunctions, where each conjunction represents a segment, it eliminates all duplicated conjunctions; it substitutes each pair of conjunctions α, β whose extensions $ext(\alpha), ext(\beta)$ partially overlap, with the minimum number of conjunctions $\gamma_1, ..., \gamma_n$ such that:

$$ext(\gamma_1) \cup ... \cup ext(\gamma_n) = (ext(\alpha) \cup ext(\beta)) - (ext(\alpha) \cap ext(\beta)).\diamond$$

Given the previous definitions, we can now define the version of GRA for spatial data.

Definition 9. *(GRA for Spatial Data)*: Given the logical theory Φ_P and the transformation operators: Bnd, Int, Buf_r, $IsLin$, $IsPoly$ and Ar, the version of GRA for spatial applications is defined as follows:

$$GRA_{spatial} = GRA(\Phi_P, \mathcal{F}) \quad \text{where} \quad \mathcal{F} = \{Bnd, Int, Buf_r, IsLin, IsPoly, \mathrm{Ar}\}.\diamond$$

Let us now consider the two approaches discussed in Section 4.1 to see how they can be mapped into $GRA_{spatial}$.

E1. In this approach, which does not impose any structure on spatial data, a set of spatial objects in E^2 can be represented as a generalized relation R of $GRA_{spatial}$ with schema $\alpha(R) = \{X, Y\}$.

E2. An example of this approach is given by the structure which describes a plane subdivision. A *plane subdivision* Σ is described by the triple (V, E, F), where F is the set of the *faces* of Σ, E is the set of the *edges* of Σ, and V is the set of the *vertices* of Σ. For the definition of the relationships existing among V, E and F, see [5].
A plane subdivision $\Sigma = (V, E, F)$ can be represented in $GRA_{spatial}$ by using three generalized relations: R_V, R_E, R_F over variables X and Y, where R_V represents *vertices*, R_E represents *edges* and R_F represents *faces*.

The properties of a plane subdivision which relate faces, edges and vertices must be represented in GRA as integrity constraints. In this paper, we focus on the first case E1, which is anyway more general than E2.

5 Spatial Queries

The large range of applications which use spatial data generates different functional requirements for spatial query languages. Therefore, a large number of operations are considered basic tools for a spatial database. This is also the reason why no completeness results can be given for any query language that is designed for spatial applications and the complexity of almost all of them is higher than the complexity of the classical relational algebra.

All spatial queries presented in the following must be considered as classical queries (as defined in [4]). Thus, we do not consider generic spatial queries with respect to geometric or topological transformations, as they have been defined in [19].

The introduction of spatial information in databases requires to enrich query languages in order to take into consideration the characteristics of this new type of data within query predicates. Queries are influenced in particular by the following features of spatial data:

1. *spatial relationships with the embedding space*: they derive from the position of the object in the space;
2. *spatial relationships with other objects in the same space*: we classify in this group topological relationships (deriving from the introduction of a concept of boundary), metric relationships (deriving from the introduction of a distance concept in the embedding space) and direction relationships (requiring the definition of some reference vector in the embedding space);
3. *spatial properties*: they are all various measurements (area, perimeter, volume, etc.) which describe the size of spatial objects and all different classifications of their shapes.

From these features, several classes of queries can be derived. In the following, we propose some of them. For each class of queries, we present an example and we translate it into the corresponding $\mathrm{GRA}_{spatial}$ expression.

Other examples are presented in Tables 4, 5 and 6. Each query is described by assigning a textual description and the mapping to $\mathrm{GRA}_{spatial}$. In the description of queries we refer to two sets of spatial objects, which are represented in $\mathrm{GRA}_{spatial}$ by two generalized relations R and S, where $\alpha(R) = \alpha(S) = \{X, Y\}$. We assume that the Cartesian product between R and S renames variables X and Y in the schema of S in X' and Y'. In the tables, $\circ$ denotes the function composition operator. For a detailed description of spatial queries, refer to [5, 8, 9, 21, 22].

5.1 Queries Based on Relationships with the Embedding Space

1. POINT INTEFERENCE QUERIES: These queries retrieve the set of spatial objects that are somehow related to a single given point P.
   EXAMPLE: *Point Intersection*
   DESCRIPTION: "Select all spatial objects in R that contain a given point

Type	Query	GRA expr.	Conditions
Point interference queries			
POINT INTERSECTION	select all spatial objects in R that contain a given point pt	$\sigma^s_{(P,\supseteq)}\ (R)$	$P = map(pt)$
Range queries			
RANGE INTERSECTION	select all spatial objects in R that intersect the region of the space identified by a given rectangle rt	$\sigma^s_{(P,(\bowtie\neq\emptyset))}\ (R)$	$P = map(rt)$
RANGE CONTAINMENT	select all spatial objects in R that are contained in the region of the space identified by a given rectangle rt	$\sigma^s_{(P,\subseteq)}\ (R)$	$P = map(rt)$
RANGE CLIP	calculate all spatial objects obtained as intersection of each object in R with a rectangle rt	$\sigma^s_{(\neq\emptyset)}(\sigma^t_P\ (R))$	$P = map(rt)$
RANGE ERASE	calculate all spatial objects obtained as difference of each object in R with a rectangle rt	$\sigma^s_{(\neq\emptyset)}(R \setminus^t P)$	$P = map(rt)$

Table 4. Examples of queries based on relationships with the embedding space

pt".
GRA EXPRESSION:

$$\sigma^s_{(P,\supseteq)}\ (R)$$

where $P = map(pt)$.
COMMENTS: Condition $(P, \supseteq)$ selects all d-generalized tuples whose extension contains the extension of the constraint corresponding to pt.

2. RANGE QUERIES: These queries retrieve the set of spatial objects that are somehow related to a range, which is represented by a hyper-rectangle in the embedding space.
EXAMPLE: *Range Containment*
DESCRIPTION: "Select all spatial objects in R that are completely contained in a given hyper-rectangle rt of the embedding space".
GRA EXPRESSION:

$$\sigma^s_{(P,\subseteq)}\ (R)$$

where $P = map(rt)$.
COMMENTS: Similar to Point Intersection.

5.2 Queries Based on Relationships with Other Objects in the Same Space

1. TOPOLOGICAL QUERIES: These queries retrieve the set of spatial objects that have a given topological relationship with a given spatial object *sp* of the database.
   EXAMPLE: *Adjacent Selection*
   DESCRIPTION: "Select all spatial objects in R that are adjacent to a spatial object *sp*".
   GRA EXPRESSION:
   $$\sigma^s_{c_1}(\sigma^s_{c_2}(R))$$
   where $P = map(sp),\ c_1 = (Int(P), (\bowtie = \emptyset)),\ c_2 = (Bnd(P), (\bowtie \neq \emptyset))$.
   COMMENTS: Selection $\sigma^s_{c_2}(R)$ retrieves from R all the d-generalized tuples having a non-empty intersection with the constraint representing the boundary of *sp*. Selection $\sigma^s_{c_1}$, applied to the resulting relation, retrieves all the d-generalized tuples having an empty intersection with the constraint representing the interior of *sp*. Thus, the retrieved tuples exactly correspond to objects adjacent to *sp*.
2. METRIC QUERIES: These queries retrieve the set of spatial objects that have a given metric relationship with a given spatial object *sp* of the database.
   EXAMPLE: *Distance Selection*
   DESCRIPTION: "Select all spatial objects in R that are within 10 Km from a spatial object *sp*".
   GRA EXPRESSION:
   $$\sigma^s_{(Buf_{10(P),(\bowtie \neq \emptyset))}}(R)$$
   where $P = map(sp)$.
   COMMENTS: Function $Buf_{10}(P)$ generates a constraint representing the object in the space containing all the points with a distance less than 10 Km from *sp*. The use of this function transforms this query into a topological query. The selection condition allows the retrieval of all d-generalized tuples of R having a non-empty intersection with this object.
3. SPATIAL JOIN QUERIES: These queries retrieve all pairs of spatial objects that satisfy a given spatial relation.
   EXAMPLE: *Spatial Join Intersection Based*
   DESCRIPTION: "Generate all pairs of spatial objects $(r, s), r \in R, s \in S$, such that r intersects s".
   GRA EXPRESSION:
   $$\sigma^s_c(R \times S)$$
   where $c = (\pi_{[X,Y]}, g, (\bowtie \neq \emptyset)),\ g = \varrho_{[X'|_X, Y'|_Y]} \circ \pi_{[X',Y']}$.
   COMMENTS: The Cartesian product generates a new relation with four variables X, Y, X', Y', containing all pairs of objects (r, s), $r \in R$ (identified by variables X, Y) and $s \in S$ (identified by variables X', Y'). The selection allows the retrieval of the pairs in which r (obtained by projecting the tuple on X, Y) has a non-empty intersection with s (obtained by projecting the

Type	Query	GRA expr.	Conditions
Topological queries			
ADIACENT SELECTION	select all spatial objects in R that are adjacent to a spatial object sp	$\sigma^s_{c_1}(\sigma^s_{c_2}(R))$	$P = map(sp)$ $c_1 = (Int(P), (\bowtie = \emptyset))$ $c_2 = (Bnd(P), (\bowtie \neq \emptyset))$
CONTAINED SELECTION	select all spatial objects in R that contain a spatial object sp	$\sigma^s_{(P,\supseteq)}(R)$	$P = map(sp)$
ON BORDER SELECTION	select all spatial objects in R that are contained in the boundary of sp	$\sigma^s_{(Bnd(P),\subseteq)}(R)$	$P = map(sp)$
Metric queries			
DISTANCE SELECTION	select all spatial objects in R that are within 10 Km from a spatial object sp	$\sigma^s_c(R)$	$P = map(sp)$ $c = (Buf_{10}(P), (\bowtie \neq \emptyset))$
Spatial join queries			
SPATIAL INTERSECTION	generate all spatial objects that are intersection between one object in R and one object in S	$R \bowtie S$	
SPATIAL JOIN (intersection based)	generate all pairs of spatial objects (r, s) $r \in R$, $s \in S$, such that r intersects s	$\sigma^s_c(R \times S)$	$c = (\pi_{[X,Y]}, g, (\bowtie \neq \emptyset))$ $g = \varrho_{[X'\mid_X, Y'\mid_Y]} \circ \pi_{[X',Y']}$
SPATIAL JOIN (adjacency based)	generate all pairs of spatial objects (r, s), $r \in R, s \in S$, such that r is adjacent to s	$\sigma^s_{c_1}(\sigma^s_{c_2}(R \times S))$	$c_1 = (Int \circ \pi_{[X,Y]}, Int \circ g, (\bowtie = \emptyset))$ $c_2 = (Bnd \circ \pi_{[X,Y]}, Bnd \circ g, (\bowtie \neq \emptyset))$ $g = \varrho_{[X'\mid_X, Y'\mid_Y]} \circ \pi_{[X',Y']}$
SPATIAL JOIN (distance based)	generate all pairs of spatial objects (r, s) $r \in R$, $s \in S$, such that r is within 10 Km from s	$\sigma^s_c(R \times S)$	$c = (\pi_{[X,Y]}, Buf_{10} \circ g, (\bowtie \neq \emptyset))$ $g = \varrho_{[X'\mid_X, Y'\mid_Y]} \circ \pi_{[X',Y']}$

Table 5. Examples of queries based on relationships with other objects in the same space

Type	Query	GRA expr.	Conditions
Measure queries			
SIZE SELECTION	select all spatial objects in R that have an area greater that 100 Km^2.	$\pi_{[X,Y]}(\sigma^s_P(AT_{Ar}(R)))$	$P = (A > 100)$
SHAPE SELECTION	select all spatial objects in R that are polygons	$\sigma^s_{(\neq\emptyset)}(AT_{IsPoly}(R))$	

Table 6. Examples of queries based on spatial properties

tuple on X', Y'). In order to check the intersection, r and s must be represented with the same variables. For this reason, function g renames the variables of s, that become X, Y[11].

5.3 Queries Based on Spatial Properties

1. MEASURE QUERIES: These queries retrieve all spatial objects that have some given properties concerning their size or their shape.
   EXAMPLE: *Shape Selection*
   DESCRIPTION: "Select all spatial objects in R that are polygons."
   GRA expression:
   $$\sigma^s_{(\neq\emptyset)}(AT_{IsPoly}(R))$$
   COMMENTS: The application of function AT_{IsPoly} to R, for each d-generalized tuple in R, returns: the same tuple, if it represents a polygon; an inconsistent d-generalized tuple, otherwise. The external set selection then removes inconsistent tuples from the generated relation.

Note that not all the useful spatial queries can be expressed in $\text{GRA}_{spatial}$. Consider for example the *Adjacent Union* query that, given a set of polygons, generates a new polygon by merging the adjacent polygons that share some properties. The detection of all adjacent polygons requires a transitive closure computation that cannot be expressed in $\text{GRA}_{spatial}$. Another query that cannot be expressed in $\text{GRA}_{spatial}$ is the *Nearest Neighbor* query that, given a set of spatial objects, returns the nearest one to a query point or to a query object. In this case, the query explicitly needs the use of aggregate functions, not available in $\text{GRA}_{spatial}$.

6 Conclusions and Future Work

Constraint databases model infinite sets of tuples, finitely represented by constraints. In this paper, we have presented a constraint relational model and a

[11] Note that the used operator $\varrho_{[X'|_X, Y'|_Y]} \circ \pi_{[X',Y']}$ can be defined as a function contained in $\mathcal{F}$. In this case, the user does not have to deal with renaming.

family of constrained languages, GRA($\Phi, \mathcal{F}$). Each language in this family is obtained by specifying a decidable logical theory Φ and a set $\mathcal{F}$ of functions, each representing an operator considered useful in a given application domain. Thus, the languages we introduce consist of an application independent set of operators and of a set of application dependent ones. Some operators are classified into set and tuple operators, with respect to the way in which d-generalized tuples are used.

We have then analyzed the issues concerning the modeling of spatial data in the proposed constraint data model and the application of the proposed language to spatial queries modeling [5, 8, 9, 21, 22].

Much work is still to be done. The definition of a declarative calculus equivalent to the proposed algebra is a first important aspect. This calculus should be equivalent to the one proposed in [13], but more suitable from a user point of view. The definition of an application dependent integrity constraint theory is another interesting topic. A related issue is the definition of a transaction language for generalized databases. Finally, the development of optimization methods for generalized databases [1, 3] is included in future work.

References

1. A. Brodsky, J. Jaffar, and M.J. Maher. Toward Practical Constraint Databases. In *Proc. of the 19th VLDB Conference*, pages 567–579, Dublin, Ireland, 1993.
2. A. Brodsky and Y. Kornatzky. The $\mathcal{L}_{yri}\mathcal{C}$ Language: Querying Constraint Objects. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, 1995.
3. A. Brodsky, C. Lassez, J.L. Lassez, and M. Maher. Separability of Polyhedra and a New Approach to Spatial Storage. In *Proc. of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, San Jose, California, 1995.
4. A.K. Chandra and D. Harel. Computable Queries for Relational Data Bases. *Journal of Computer and System Sciences*, 21:156–178, 1980.
5. L. De Floriani, P. Marzano, and E. Puppo. Spatial Queries and Data Model. In A.U. Frank, I. Campari, U. Formentini, editors, *Spatial Information Theory: a Theoretical Basis for GIS*, volume 716 of *Lectures Notes in Computer Sciences*, Springer-Verlag, pages 123–138, September 1993.
6. M. J. Egenhofer and J. R. Herring. A Mathematical Framework for the Definition of Topological Relationships. In *Proc. of the 4th Symp. on Spatial Data Handling*, pages 803-813, 1990.
7. M.J. Egenhofer. Reasoning about Binary Topological Relations. In *Proc. of the 2nd Symp. on Advances in Spatial Databases*, pages 143-160, 1991.
8. M. Gargano, E. Nardelli, and M. Talamo. Abstract Data Types for the Logical Modeling of Complex Data. *Information Systems*, 16(6):565–583, 1991.
9. R.H. Gueting and M. Schneider. Realm-Based Spatial Data Types: The ROSE Algebra. *Fernuniversität Hagen* Informatik-Report 141, 1993. To appear in the *VLDB Journal*.
10. M.R. Hansen, B.S. Hansen, P. Lucas, and P. van Emde Boas. Integrating Relational Databases and Constraint Languages. *Computer Languages*, 14(2):63–82, 1989.

11. P. Kanellakis. Elements of Relational Database Theory. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, Chapter 17, 1990.
12. P.C. Kanellakis and D.Q. Goldin. Constraint Programming and Database Query Languages. Technical Report CS-94-31, Brown University, Providence, USA, 1994.
13. P.C. Kanellakis, G.M. Kuper, and P.Z. Revesz. Constraint Query Languages. *Journal of Computer and System Sciences*, to appear. See also *Proc. of the 9th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 299–313, Dever, Colorado, April 1990.
14. M. Kifer, W. Kim, and Y. Sagiv. Querying Object-Oriented Databases. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 393–402, 1992.
15. M. Koubarakis. Representation and Querying in Temporal Databases: the Power of Temporal Constraints. In *Proc. of the Int. Conf. on Data Engineering*, pages 327–334, 1993.
16. J.L. Lassez. Querying Constraints. In *Proc. of the 11th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 288–298, Nashville, Tennessee, April 1990.
17. J. Nievergelt. "7 +− 2 Criteria for Assessing and Comparing Spatial Data Structures. In *Proc. First Symposium Large Spatial Databases*, pages 5-25, Santa Barbara, California 1989.
18. J. Paredaens. Spatial Databases, The Final Frontier. In G. Gottlob and M. Vardi, editors, volume 893 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994.
19. J. Paredaens, J. Van den Bussche, and D. Van Gucht. Towards a Theory of Spatial Database Queries. In *Proc. of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 279–288, Minneapolis, Minnesota, USA, 1994.
20. F. P. Preparata and M.I. Shamos. *Computational Geometry - an Introduction*, Springer Verlag, New York, 1985.
21. M. Scholl and A. Voisard. Thematic Map Modeling. In *Proc. of the Symp. on the Design and Implementation of Large Spatial Databases*, 1989, pages 167–190.
22. P. Svensson. GEO-SAL: a Query Language for Spatial Data Analysis. In *Proc. of the 2nd Symp. on Advances in Spatial Databases*, pages 119-140, 1991.

This article was processed using the LaTeX macro package with LLNCS style