Hierarchical Optimization of Linear Constraint Processing

Stéphane Grumbach[1], Philippe Rigaux[2], and Pierangelo Veltri[1]

[1] INRIA Rocquencourt, BP 105, F-78153 Le Chesnay, France
{Stephane.Grumbach, Pierangelo.Veltri}@inria.fr
[2] Cedric/CNAM, 292 rue St Martin, F-75141 Paris Cedex 03, France
rigaux@cnam.fr

Abstract. We consider the problem of solving a *large* number of *simple* systems of constraints. This problem occurs in the context of databases with very large collections of data while the constraints are over a small number of variables. The methodology we develop is based on a hierarchical evaluation of the constraints which are first simplified, and replaced by constraints approximating the initial ones.
We focus on systems of linear constraints over the reals, which model spatial objects, and consider both geometric and topological approximations, defined with very simple constraints. We show that these constraints can be used either to solve the initial systems, or at least to filter out unsatisfiable systems.
More generally, we consider the manipulation of the spatial objects with first-order queries. We show how the queries can be evaluated by taking advantage of the approximations. In general, it is undecidable if a query can be completely answered on approximated data. The main contribution of the paper is the development of a set of rewriting rules, that allow the transformation of queries into equivalent ones that make use of the approximation. As usual with optimization problems, an optimal solution is out of reach, but the rules are optimal for individual operations of relational algebra.
The efficiency of the technique is important for the range of applications considered, where the probability of filtering from the approximated data is very high. The technique is illustrated on a practical example.

1 Introduction

Constraints are used in various areas of computer science, such as for instance logic programming, combinatorial optimization, and databases ([KKR90], [KLP00]). Improving the efficiency of constraint processing is a challenge with immediate impact in the applications. In this paper, we consider the problem of solving a *large* number of *simple* systems of constraints, which occurs in the context of databases. The emphasis is therefore on the number of systems we want to solve, versus the complexity of each individual system. The problem is motivated by applications such as those involving spatial data.

Spatial databases contain collections of geometric objects, such as polygons in a 2d space. The manipulation of these objects is costly because of the intricacy of the geometry. To guaranty efficient access to the objects in a database, special data structures (indexes) are designed, which ensure a logarithmic time search of objects. In the case of spatial data, the logarithmic time cannot be guaranteed, but can be reached in practice through spatial indexes such as the R-tree [Gut84] and its variants [SRF87,BKSS90]. Spatial indexes are not designed directly on the geometry of the objects, but on their minimal bounding boxes, that is the minimal rectangle containing them, which constitutes an approximation that can be used to help processing the initial objects.

The polygons can be easily represented by systems of linear inequalities over the reals. See Fig. 1 for an example. The minimal bounding boxes on the other hand can be represented by systems of univariate constraints, which are of course much easier to process than the general linear constraints. The use of constraints to represent geometric data, for databases has been primarily advocated in [KKR90]. Since then, the constraint database area has grown ([KKR95],[KLP00]) with a large emphasis on theoretical results related to the expressive power and complexity of query languages ([GST94], [GS97], [GRS99]).

$$
\begin{array}{lll}
 & y \leqslant x & (1) \\
\wedge & x \leqslant 5 & (2) \\
\wedge & y \geqslant 1 & (3) \\
\wedge & 3y + x \leqslant 24 & (4)
\end{array}
$$

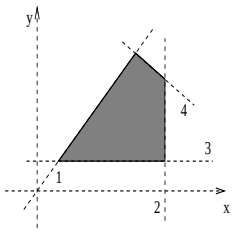


Fig. 1. A system of constraints and its associated geometry $\mathcal{P}$

We develop a strategy to solve the systems of constraints, which is based on a hierarchical evaluation. Constraints are initially simplified, and the simplified constraints are then processed. The evaluation over simplified constraints can be done quite efficiently, and allows to determine in many cases whether the initial system is satisfiable or not. For the data that qualify to this *filtering step*, the initial constraints need generally to be processed. In practice, the process allows to reduce the application of the costly constraint solving operations to a small subset of the input, and results in drastic optimization.

In the present work, we consider a hierarchical evaluation technique, based on three classes of constraints, (i) *linear constraints* over the reals, (ii) *box constraints*, which are univariate constraints, and (iii) *topological constraints*.

We assume that the objects in the database are stored with both a *geometry*, which is a polyhedron represented using linear constraints, and a *bounding box*, which is a rectangular approximation of the geometry. This is a standard assumption in spatial information systems. Observe that the bounding-box approximation is *conservative*: any point in the geometric object belongs to the bounding-box as well.

Moreover we assume that topological relations between the objects are stored in the database. Egenhofer and Franzosa [EF91] have proposed binary topological relations among pairs of spatial regions, based on the intersections of their topological interiors, boundaries, and exteriors. These mutually exclusive relations are: *overlaps, disjoint, equal, adjacent, contains, covers*, and the inverse of the last two. Three of these relations play a fundamental role for constraint processing and query evaluation: *overlap, disjoint*, and *inside*. We assume that the database contains these binary relations, which store pairs of database objects that satisfy the corresponding topological condition.

In databases, objects are manipulated through query languages. The basic operations require the processing of constraints to produce an output. The goal of this paper is to show how the box constraints and the topological constraints can be involved in an efficient query processing of general linear constraints, using a hierarchical algorithm as outlined above.

Our contribution is the design of rewriting rules for algebraic queries which introduces a preliminary evaluation phase on box constraints and topological constraints. We prove that the rewriting system is sound and optimal for the basic operations of relational algebra. It is sound in the sense

that the rewritten operation is equivalent to the initial one, and optimal in the sense that no more filtering could have been done using the approximation.

The present techniques are under implementation in the DEDALE system [GRS98], a database system based on the linear constraint model. The data model of DEDALE provides non-first normal form relations with one level of nesting. Alphanumeric information can be easily represented in unnested attributes while one level of set nesting can be used for spatial data. Such a model has also been proposed in [BBC97].

The paper is organized as follows. In the following section, we illustrate the principle of hierarchical processing of constraints. In Section 3, we define formally the systems of constraints we consider, and the first-order manipulations. In Section 4 we present a manipulation language for collections of spatial data. The rewriting rules for hierarchical constraint processing are presented in Section 5.

2 Constraint processing

The focus of the present work is on linear constraints over the reals. Linear constraints are of the form:

$$\sum_{i=1}^{p} a_i\, x_i = a_0, \qquad \sum_{i=1}^{p} a_i\, x_i \leqslant a_0$$

where the x_i's denote variables and the a_i's are rational constants, and we don't consider irrational parameters. Fig. 1 shows for instance a system of constraints and its corresponding polygon $\mathcal{P}$.

We study systems of linear constraints over d variables for some fixed d which corresponds to the dimension of the space. To catch the intuition, we fix $d = 2$ in examples. Each such system defines a convex polyhedron in the real space.

We consider the manipulation of these polyhedra through various operations, such as geometric transformations: e.g.,projection, and set-theoretic operations: e.g.,intersection. These operations are evaluated by processing the constraints defining the objects. The complexity of their evaluation depends upon the number of constraints involved, and upon the global dimension.

- **projection**, π, corresponds to the elimination of variables from the system of constraints. The cost of eliminating *one* variable from a system of n constraints over d variables is linear for $d = 1$, in $O(n \log n)$ for $d = 2$, and quadratic in general using the Fourier method [Sch86]. The cost of eliminating k variables is then in $O(n^{2^k})$ in general. It can also be done more subtly in $O(k2^{2^d} n^4)$.
- **intersection**, $\cap$. The intersection of two objects can be defined by the conjunction of their systems of constraints. From a geometric point of view, the computation of the intersection corresponds in a simplification of the resulting system. Its complexity increases with the number of variables. In dimension 2 it can be performed in $O(n \log n)$ by a half-plane intersection algorithm [PS85].

If the operations need to be applied to large collections of objects, the constraint processing phase should be optimized. This can be achieved by considering an approximated representation of the objects with simpler constraints which can be evaluated much more efficiently. We thus introduce a class of simple constraints called *box constraints*.

Definition 1 A *box constraint* is a linear constraint with a unique variable, of the form $x = a$, or $x \leqslant a$, for a variable x and a constant a.

Box constraint processing is extremely simple. The resolution of such systems of box constraints can be done in time linear in the number of constraints independently of the dimension.

Box constraints can only define *rectangular polyhedron*, that is polyhedron whose faces are parallel to the axis. Systems of linear constraints can be approximated by systems of box constraints using a geometric construction.

Definition 2 Given a polyhedron, its *minimal bounding box*, is the smallest rectangular polyhedron containing it.

The *geometric approximation* of a system of linear constraints representing a polyhedron is defined as the box system of the minimal bounding box of the polyhedron.

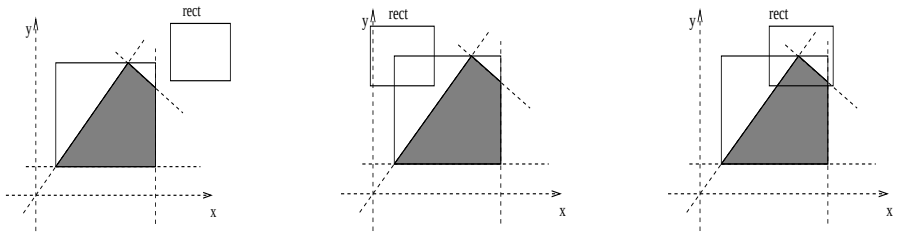
As explained above, the geometric approximation of systems of linear constraints can be used to increase the efficiency of linear constraint processing. Moreover, in some cases, the result of an operation can be computed directly from the geometric approximation. Consider for instance the projection over one variable. The geometric approximation of the system of Fig. 1 is given by

$$1 \leqslant x \leqslant 5 \wedge 1 \leqslant y \leqslant 6$$

The projection of the object $\mathcal{P}$ of Fig. 1 on either x or y can be obtained directly from the geometric approximation:

$$\pi_x \mathcal{P} = \pi_x Box(\mathcal{P})$$

Consider now the intersection. Two objects have a non empty intersection only if their geometric approximations have a non empty intersection. For instance in Fig. 2.c, the rectangle *rect* intersect the polygon $\mathcal{P}$, and their approximations intersect too. If the approximation of $\mathcal{P}$ and *rect* do not intersect, (Fig. 2.a), then the result (empty) can be obtained directly from the geometric approximation. If object approximations intersect the result can nevertheless be empty (Fig. 2.b).



(a) (b) (c)

Fig. 2. Intersection with a rectangle

With the bounding box approximation, one of the two following cases occurs:

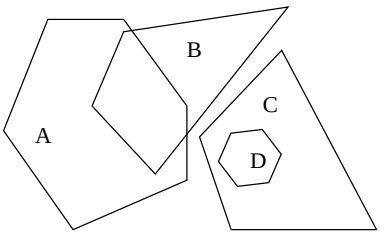
1. The minimal bounding box of the two objects intersect, the two objects are candidates and the value needs to be computed;
2. The minimal bounding box of the two objects do not intersect, the result (empty) is obtained with no computation.

Let us consider now how the topology can be used to increase the efficiency of linear constraint processing. We consider objects o which have identifiers o_id, a geometry denoted by $o.geo$, and a bounding box denoted by $o.box$ defined with linear constraints.

Definition 3 A *topological constraint* is one of the following expressions: *disjoint(o,o')*, *inside(o,o')*, *overlap(o,o')* where o and o' are objects.

Definition 4 Let o and o' be objects with geometry $o.geo$, and $o'.geo$. The topological predicates *disjoint*, *inside*, *overlap*, are defined as follows:

- *disjoint(o,o')* is true if and only if $o.geo \cap o'.geo = \emptyset$.
- *inside(o,o')* is true if and only if $o.geo \cap o'.geo = o.geo$.
- *overlap(o,o')* is true if and only if $o.geo \cap o'.geo \neq \emptyset$ and $o.geo \cap o'.geo \neq o.geo$ and $o.geo \cap o'.geo \neq o'.geo$.



overlap	
A	B
B	A

inside	
D	C

disjoint	
A	C
A	D
B	C
B	D
C	A
C	B
D	A
D	B

Fig. 3. Geometric objects and their topology

Consider now the intersection between two objects. In the case of Fig. 3, *disjoint(B,C)* is true, and the empty result can be obtained directly from the topological relation. If we had considered the intersection between the two mbbs, we would need to evaluate the intersection between the two objects to get the results. Thus the topological filter is in this case more efficient than the mbb.

For the intersection between two objects o and o', one of the following cases occurs with the topological approximation:

1. *disjoint(o,o')*: the filtering step is sufficient, the result is empty;
2. *overlap(o,o')*: the pair of objects is accepted and intersection value needs to be computed;
3. *inside(o,o')*: the object is accepted and the result, o, is obtained with no computation.

3 Geometric data modeling

In this section, we first describe the general framework for the representation of geometric pointsets using linear constraints, and then define a simple manipulation language based on first-order logic.

Geometric data is modeled as infinite sets in the real space. For example, if $\mathbb{Q}$ is the universe of the real number, a polygon in the plane is seen as the infinite set of points of $\mathbb{Q}^2$ inside its frontier, and a 3-dimensional pyramid is seen as the infinite set of points of $\mathbb{Q}^3$ inside its facets.

At this *abstract level*, spatial data consists of infinite relations over the real numbers $\mathbb{Q}$. These infinite relations can only be described and manipulated through a finite representation. Following the trends of constraint databases [KKR90], we use first-order logic to represent the relations of interest, and define a *symbolic level* of representation. We distinguish clearly between the *intensional* representation of a relation at the symbolic level, and its *extensional* interpretation at the abstract level.

We consider linear constraints in the first-order language $\mathcal{L} = \{\leqslant, +\} \cup \mathbb{Q}$ over the structure $\mathcal{Q} = \langle \mathbb{Q}, \leqslant, +, (q)_{q\in\mathbb{Q}} \rangle$ of the linearly ordered set of the real numbers with rational constants and addition. Linear constraint relations are defined as follows.

Definition 5 Let $S \subseteq \mathbb{Q}^k$ be a k-ary relation. The relation S is a linear constraint relation if there exists a formula $\varphi(x_1, \ldots, x_k)$ in $\mathcal{L}$ with k distinct free variables $x_1, \ldots, x_k$ (called a *representation* of S) such that:

$$\mathcal{Q} \models \forall x_1 \cdots x_k (S(x_1, \ldots, x_k) \leftrightarrow \varphi(x_1, \ldots, x_k))$$

The class of linear constraint relations constitutes a rather drastic restriction over the class of all infinite relations over $\mathbb{Q}^k$. This restriction is necessary to ensure reasonable query complexity and is sufficient to encompass spatial data in computational geometry, GIS, etc. as it has already been widely demonstrated in the literature ([GST94]).

More drastic restrictions have been traditionally imposed in constraint databases [KKR90]. The first one is to represent relations by quantifier free formulae. This restriction is harmless, since for each formula, there is an equivalent quantifier free formula, which moreover can be constructed effectively. Therefore each spatial object can be represented by a quantifier free formula. The rationale for the quantifier free representation is the resulting reasonable query complexity. The second restriction enforces the formula to be in disjunctive normal form (DNF). This restriction is harmless as well, since each formula can be put in DNF, and again the motivation is complexity.

A DNF formula $\varphi \equiv \bigvee_{i=1}^{k} \bigwedge_{j=1}^{\ell_i} \varphi_{i,j}$ is seen as a *symbolic relation*, that is a finite set of *symbolic tuples*, which are conjunctions of atomic formulae $\varphi_{i,j}$

$$\left\{ t_i \;\middle|\; 1 \leqslant i \leqslant k,\ t_i = \bigwedge_{j=1}^{\ell_i} \varphi_{i,j} \right\}$$

The representation in terms of DNF formulae enforces a specific view of spatial objects. Indeed, each relation is the union (disjunction) of a finite set of convex objects (symbolic tuples). In terms of 2-dimensional applications, this implies for example that polygons must be decomposed into convex component polygons (convexified).

3.1 Poinset manipulation language

We now consider queries which can be expressed as first-order formulae in the language $\mathcal{L}$ over $\mathcal{Q}$. If s is a schema, each formula φ in $\mathcal{L} \cup s$ with free variables $x_1, \ldots, x_n$ ($n \geqslant 0$) defines a *query over* s, mapping instances I of s to n-ary relations.

Suppose $q = \{(x_1, \ldots, x_n) \mid \varphi\}$ is a query over s, and I is an instance of s. Since each relation R in I can be defined by a quantifier-free formula in $\mathcal{L}$, and φ is a formula in $\mathcal{L} \cup s$, we can replace in φ each occurrence of the relation symbol $R \in s$ by a formula defining R. The resulting formula φ' is a formula in $\mathcal{L}$ with no reference to relation symbols in s, which defines the *answer to the query* q *on* I, denoted by $q(I)$.

The query evaluation is based on an *algebra*, equivalent to first-order logic. The algebra consists of the following operations: Cartesian product, $\times$, projection, π, union, $\cup$, set difference, $-$. The algebra operations, whose effect is described below, are performed on sets of symbolic tuples. Let R_1 and R_2 be two relations, and respectively e_1 and e_2 be sets of symbolic tuples defining them.

1. $R_1 \times R_2 = \{t_1 \wedge t_2 \mid t_1 \in e_1, t_2 \in e_2\}$.
2. $\sigma_F(R_1) = \{t_1 \wedge F \mid t_1 \in e_1\}$ where F is any atom of $\mathcal{L}$.

3. $\pi_{\overline{x}}\ R_1$ is computed using the algorithm of quantifier elimination[1].
4. $R_1 \cup R_2 = e_1 \cup e_2$.
5. $R_1 - R_2 = \{t_1 \wedge t_2 \mid t_1 \in e_1, t_2 \in (e_2)^c\}$, where e^c is the set of tuples or disjuncts of a DNF formula corresponding to $\neg e$.
6. $simplify(R_1)$, eliminates redundancies and detects inconsistencies.

The symbolic operations above are well defined in the sense that their effect on the intensional definition of sets corresponds exactly to the semantics of the corresponding relational operators from the relational algebra over the possibly infinite extension of the sets. The relational intersection and join are definable with the symbolic Cartesian product. The sets of variables are different in these three operations. For the Cartesian product, the variables of the two relations are disjoint, they are similar in the case of the intersection, and with a non empty intersection in the case of the join and the selection. The *simplify* operator, given a conjunction of constraints, eliminates redundancies, and detects inconsistencies. Simplification is necessary for checking the satisfiability of formulae.

3.2 Queries using approximation

We consider here the possibility for a query to be answered using the approximation either directly or by rewriting the query in a new query over the approximation. We first consider negative results. We say that a query can be *directly answered* on an approximation if for all inputs, the query on the approximation of the input produces the same result as the query on the initial input.

Proposition 1 It is undecidable if a first-order query can be directly answered (i) on the geometric approximation, (ii) on the topological approximation, or (iii) on both approximation.

We say that a query Q can be *answered* on an approximation if there is a query Q', such that for all inputs, the query Q' on the approximation of the input produces the same result as the query Q on the initial input.

Proposition 2 It is undecidable if a first-order query can be answered (i) on the geometric approximation, (ii) on the topological approximation, or (iii) on both approximation.

4 Manipulation of data collections

This model is the one adopted in DEDALE [GRS98] and advocated in [BBC98].

We consider collections of real-world entities whose description include - but is not limited to - a set of points such as in geographic applications for instance, where an abstract description of the real-world distinguishes different classes of entities on the earth surface: *buildings*, *roads*, *parcels*. A class of entity is described by several properties (usually denoted *attributes* in the DB terminology), for instance *owner* and *ground-occupancy* for *parcels*. If the application aims at manipulating the location of these objects, a specific attribute is the *spatial* description, i.e, an infinite set of points. Geographic modeling usually distinguishes several types of geometric data: *points*, *lines*, *polygons*, *regions*, etc. We shall use the more abstract point of view of symbolic relations, as defined above.

So we need a data model allowing the representation and manipulation of classical attributes (*owner* for instance) as well as geometric attributes. To this end we store data in non-first normal form relations [Ull88] to allow the manipulation of infinite sets of points as first class citizens. One level of set nesting is sufficient to achieve this goal.

[1] In the linear case this is done by the Fourier-Motzkin elimination method [Sch86].

Fig. 4 shows an example of a collection schema describing the class *Parcel* and its instance: *owner* and *ground-occupancy* are represented as classical atomic values, while the nested attributes *geo* is a symbolic relation represented as FO formulae.

$Parcel = \{[$ owner : *string*
ground-occupancy : *string*
geo : *pointset*
$]\}$

owner	ground-occupancy	geo
John	Pasture	$(1230x + 134y \leqslant 23$ $\wedge(3x - 2y \leqslant 49 \wedge 23x + 2y \geqslant 134$ $\ldots \wedge 3x - 467y \geqslant 265)\ldots$

Fig. 4. Schema and instance of the collection *Parcel*: the nested attributes represent pointsets

4.1 Collection manipulation language

Collections can be manipulated with relational algebra operators. Moreover in order to query geometric data, we introduce a MAP operator to apply algebraic expressions to nested pointsets.

Given a collection R, $MAP_{\lambda X.[E(X)]}\, R$ defines a collection of objects R', in which each object o of R has been replaced by $E(o)$, and $E(X)$ is a tuple of algebraic expressions, $E(X) = e_1(x), \ldots e_m(x)$.

As a simple example, consider the following *windowing query* over the collection $Parcel$: *give the owner of those parcels which cross the rectangle rect* Assuming that *rect* is represented by the constraints $\{x_{min} \leqslant x \leqslant x_{max}, y_{min} \leqslant y \leqslant y_{max}\}$, the logical expression is

$$MAP_{\lambda X[X.owner, X.geo \cap rect]}(Parcel)$$

The semantic of $X.geo \cap rect$ is the pairs (x, y) in the *geo* attribute of $Parcel$ such that (x, y) can be also found in *rect*. Since both extensions are infinite, this result is represented by linear constraints. Those objects of R whose $X.geo \cap rect = \emptyset$ are discarded. Upon the instance of Fig. 4, this yields the collection of Fig. 5.

The result features a new formula for *geo* which represents exactly those points whose coordinates satisfy both the formula that represented the initial geometry and the formula representing the rectangle: in other words the intersection on x and y.

owner	geo
John	$(1230x + 134y \leqslant 23$ $\wedge(3x - 2y \leqslant 49 \wedge 23x + 2y \geqslant 134 \ldots \wedge 3x - 467y \geqslant 265)$ $\wedge x_{min} \leqslant x \leqslant x_{max} \wedge y_{min} \leqslant y \leqslant y_{max}) \ldots$

Fig. 5. The result of $MAP_{\lambda X[X.owner, (X.geo \cap rect)]}(Parcel)$

In summary, any relational algebraic expression can be applied through the MAP operator to pointsets: this allows to express easily intersections (spatial joins), (geometric) projections, differences, complex selections, etc.

In most cases the evaluation of such queries can be done in two steps. The first one is purely symbolic: as illustrated above we just make some manipulation of the constraint representation (a trivial conjunction in the example above). In a second step it remains to test for emptiness the

resulting formula which might be unsatisfiable. From now on, we consider that in each collection there is a minimal bounding box attribute associated to each geometric value. For instance the collection schema of the class *Parcel* features an additional attribute *box*. Here is now a simple improvement of the query of the above example:

$$MAP_{\lambda X.[X.owner, rect \cap X.geo]}(\sigma_{rect \cap box \neq \emptyset}(Parcel))$$

By testing the overlapping of *rect* and *box*, we can eliminate from further consideration a large amount of parcel entities for which the geometric computation would yield an empty result. This phase is very cheap and it saves the costly algorithmic on constraints for all these obvious cases. It might happen, though, that some parcel qualify to the $rect \cap box$ test, while the actual geometric representation does not intersect *rect* (false hits).

The above query rewriting is a simple but representative example of an optimization based on bounding boxes.

5 Query rewriting

In this section we introduce some rewriting rules for the query optimization. They result in a hierarchical processing of the constraints.

During the presentation, we rely on two sample collections $R = \{[geo : pointset, box : pointset]\}$, and $R' = \{[geo : poinset, box : pointset]\}$ together with the topological relations *overlap*, *disjoint*, and *inside*. Observe that, for technical reasons, we might have to deal with *partial* bounding boxes, i.e., boxes defining a range for a strict subset of the variables (e.g., the geometry can be defined on n variables and the box on m variables with $m < n$). Furthermore, we indicate the geometric attributes of R and R' as geo_R and $geo_{R'}$. From now on, for expressions that involve more than one collection, we use the notation $attribute_{collection}$ to specify to which collection the attribute belongs to.

In the following we omit in the MAP operator attributes different from geometry and minimal bounding box (i.e., we write $MAP_{\lambda X.[X.geo \cap rect]}\, R$ for $MAP_{\lambda X.[..,X.geo \cap rect,..]}\, R$).

5.1 Rewriting of the algebraic operations

The table of Fig. 6, presents the rewriting rules using minimal bounding box for the following algebraic operators: intersection, $\cap$, difference, $-$, projection, π, selection, σ, and join, $\bowtie$. The correctness of the rewriting is proved in the Appendix.

Theorem 3 The rules 1 to 9 of table of Fig. 6 are sound and optimal.

The soundness of the rules follows from the series of lemmas 4 to 12 presented in the appendix.

The table of Fig. 7 contains the rewriting rules using the topology, for queries that involve two collection R and R'. In the hypothesis that topological relations contain pairs of object's identifiers that satisfy the topological condition, with \$1 we indicate the first attribute of the relation, and with \$2 the second one. Thus, for example in $R \bowtie_{o_id=\$1} overlap$, the join condition is on the identifier of the objects of R and on the attribute of the first column of *overlap*.

[3] Projection rewriting is correct with opportune conditions. See in the Appendix for further details

[4] In the table the filtering operations are underlined.

Op	num	query	filtered query
$\cap$	1	$MAP_{\lambda X.[X.geo \cap rect]}\ R$	$MAP_{\lambda X.[X.geo \cap rect]}\ \underline{\sigma_{box \cap rect \neq \emptyset}} R$
	2	$MAP_{\lambda X.[X.geo_R \cap X.geo_{R'}]}\ R \times R'$	$MAP_{\lambda X.[X.geo_R \cap X.geo_{R'}]}\ \underline{\sigma_{box_R \cap box_{R'} \neq \emptyset}} R \times R'$
$-$	3	$MAP_{\lambda X.[X.geo - rect]}\ R$	$MAP_{\lambda X.[X.geo - rect]}\ \underline{\sigma_{box \cap rect \neq \emptyset}} R$ $\cup\ MAP_{\lambda X.[X.geo]}(\underline{\sigma_{box \cap rect = \emptyset}} R)$
	4	$MAP_{\lambda X.[X.geo_R - X.geo_{R'}]}\ R \times R'$	$MAP_{\lambda X.[X.geo_R - X.geo_{R'}]}\ \underline{\sigma_{box_R \cap box_{R'} \neq \emptyset}}(R \times R')$ $\cup\ MAP_{\lambda X.[X.geo_R]}(\underline{\sigma_{box_R \cap box_{R'} = \emptyset}}(R \times R'))$
π^3	5	$MAP_{\lambda X.[\pi_{x_i} X.geo]}\ R$	$MAP_{\lambda X.[\pi_{x_i} X.box]}\ R$
	6	$MAP_{\lambda X.[\pi_{x_1,..x_n} X.geo]}\ R$	$MAP_{\lambda X.[\pi_{x_1,..x_n} X.box]}\ R$
σ	7	$MAP_{\lambda X.[\sigma_{x_1=a_1 \wedge .. \wedge x_k=a_k} X.geo]}\ R$	$MAP_{\lambda X.[\sigma_{x_1=a_1 \wedge .. \wedge x_k=a_k} X.geo]}\ \underline{\sigma_{P \in box}} R$
	8	$MAP_{\lambda X.[\sigma_F X.geo]}\ R$	$MAP_{\lambda X.[\sigma_F X.geo]}(MAP_{\lambda X.[X.geo, \underline{\sigma_F X.box}]}\ R)$
$\bowtie$	9	$MAP_{\lambda X.[X.geo_R \bowtie X.geo_{R'}]}\ R \times R'$	$MAP_{\lambda X.[X.geo_R \bowtie X.geo_{R'}]}\ \sigma_{box_R \cap box_{R'} \neq \emptyset} R \times R'$

Fig. 6. Basic rewriting rules using mbb[4]

op	query	filtered query
$\cap$	$MAP_{\lambda X.[X.geo_R \cap X.geo_{R'}]}\ R \times R'$	$MAP_{\lambda X.[X.geo_R \cap X.geo_{R'}]}\ R \bowtie_{o_id=\$1} overlap \bowtie_{\$2=o_id} R'$ $\cup\ MAP_{\lambda X.[X.geo_R]}\ R \bowtie_{o_id=\$1} inside \bowtie_{\$2=o_id} R'$ $\cup\ MAP_{\lambda X.[X.geo_{R'}]}\ R' \bowtie_{o_id=\$1} inside \bowtie_{\$2=o_id} R$
$-$	$MAP_{\lambda X.[X.geo_R - X.geo_{R'}]}\ R \times R'$	$MAP_{\lambda X.[X.geo_R]}\ R \bowtie_{o_id=\$1} disjoint \bowtie_{\$2=o_id} R'$ $\cup\ MAP_{\lambda X.[X.geo_R - X.geo_{R'}]}\ R' \bowtie_{o_id=\$1} inside \bowtie_{\$2=o_id} R$ $\cup\ MAP_{\lambda X.[X.geo_R - X.geo_{R'}]}\ R \bowtie_{o_id=\$1} overlap \bowtie_{\$2=o_id} R'$
$\bowtie$	$MAP_{\lambda X.[X.geo_R \bowtie X.geo_{R'}]}\ R \times R'$	$MAP_{\lambda X.[X.geo_R \bowtie X.geo_{R'}]}\ R \bowtie_{o_id=\$1} overlap \bowtie_{\$2=o_id} R'$

Fig. 7. Basic rewriting rules using topological relations

5.2 Rewriting of complex queries

We present now two practical examples of queries. A combination of the rewriting rules seen above can be used to optimize the evaluation, and, in some cases to obtain the result.

Given the collections *Lake*, *Mountain*, *River*, and *Parcel* consider the query: *find the lakes inside a mountain area, south of a given point* $P(x_p, y_p)$. This query can be expressed by:

$$MAP_{\lambda X.[\sigma_{y \leqslant y_p} X.geo_{lake}]}\ \sigma_{X.geo_{lake} \subseteq X.geo_{mountain}}(Lake \times Mountain)$$

Using the topological relation *inside*, and filtering on mbbs, the query can be rewritten as

$$MAP_{\lambda X.[X.geo_{lake}]}(MAP_{\lambda X.[X.geo_{lake}, \sigma_{y \leqslant y_p} X.box_{lake}]}(Lake \bowtie_{o_id=\$1} inside \bowtie_{\$2=o_id} Mountain))$$

In this case the results are obtained with no geometric computation. Consider now the query: *find part of parcels inside a given rectangle rect and crossed by a river*. It can be evaluated without considering the approximations as:

$$MAP_{\lambda X.[X.geo_{parcel} \cap rect]}(MAP_{\lambda X.[X.geo_{parcel}, X.geo_{parcel} \cap X.geo_{river}]}\ Parcel \times River)$$

Using the *overlap* topological relation and the mbb the query is rewritten as:

$$MAP_{\lambda X.[X.geo_{parcel} \cap rect]}\ \sigma_{box_{parcel} \cap rect \neq \emptyset}(Parcel \bowtie_{o_id=\$1} overlap \bowtie_{\$2=o_id} River)$$

where with the topological filtering we obtain *all* the parcel crossed by a river (no computation), and using the mbb we filter out all the parcels whose minimal bounding box do not intersect *rect*.

6 Conclusion

We presented in this paper a technique to evaluate large collections of simple systems of constraints by hierarchically evaluating approximated constraints. Although, the general optimization problem is undecidable, we have developed sound rewriting methods for each relational operation. For arbitrary queries, we show on examples how the rewriting is carried on. This framework captures the classical sequence of *filter step* and *refinement step* widely accepted in the spatial query processing literature since the initial work of [Ore86]. However the method is in that case restricted to the specific *overlap spatial join* operation (see for instance [BKS93,BKSS94,PD96]), equivalent to our intersection operator. Our results provide a more general and sound description of these techniques.

The modeling with constraint-based data representation and first-order query languages is of great help for the optimization problem. The results lead to efficient evaluation techniques, which were illustrated with queries on spatial databases, but can be used in other contexts. The translation of a spatial query to a simplified expression involving bounding boxes is very well suited to spatial indices which are structured collection of rectangles supporting several kinds of spatial search operations (see [GG98] for a survey).

The present techniques are under implementation in the DEDALE system [GRS98]. More work is required for the development of efficient heuristics.

References

[BBC97] A. Belussi, E. Bertino, and B. Catania. Manipulating spatial data in constraint databases. In *Intl. Conf. on Advances in Spatial Databases (SSD'97)*, pages 115–141. Springer Verlag, LNCS 1262, 1997.

[BBC98] A. Belussi, E. Bertino, and B. Catania. An extended algebra for constraint databases. In *Transaction on Knowledge and Data Engineering, vol.10(5)*, pages 686–705. IEEE, 1998.

[BKS93] T. Brinkhoff, H.-P. Kriegel, and B. Seeger. Efficient Processing of Spatial Joins Using R-Trees. In *Proc. ACM SIGMOD Symp. on the Management of Data*, 1993.

[BKSS90] N. Beckmann, H.P. Kriegel, R. Schneider, and B. Seeger. The R*tree : An Efficient and Robust Access Method for Points and Rectangles. In *Proc. ACM SIGMOD Intl. Symp. on the Management of Data*, pages 322–331, 1990.

[BKSS94] T. Brinkhoff, H.P. Kriegel, R. Schneider, and B. Seeger. Multi-Step Processing of Spatial Joins. In *Proc. ACM SIGMOD Symp. on the Management of Data*, pages 197–208, 1994.

[CGK96] J. Chomicki, D.Q. Goldin, and G. Kuper. Variable Independence and Aggregation Closure. In *Proc. ACM Symp. on Principles of Database Systems*, pages 40–48, 1996.

[EF91] M. J. Egenhofer and R. Franzosa. Point-set topological spatial relations. *Int. J. Geo. Info. Sys.*, 5(2), 1991.

[GG98] V. Gaede and O. Guenther. Multidimensional Access Methods. *ACM Computing Surveys*, 30(2), 1998.

[GRS98] S. Grumbach, P. Rigaux, and L. Segoufin. The dedale system for complex spatial queries. In *Proc. ACM SIGMOD Symp. on the Management of Data*, 1998.
`http://www-rocq.inria.fr/verso/dedale/` .

[GRS99] S. Grumbach, P. Rigaux, and L. Segoufin. On the orthographic dimension of constraint databases. In *Proc. of Intl. Conf. on Database Theory, Jerusalem*, pages 199–216. Springer Verlag, 1999.

[GS97] S. Grumbach and J. Su. *Queries with arithmetical constraints.* Invited to a special issue, 1997.

[GST94] S. Grumbach, J. Su, and C. Tollu. *Linear constraint query languages: Expressive power and complexity.* LNCS 960. Springer-Verlag, 1994.

[Gut84] A. Guttman. R-trees : A Dynamic Index Structure for Spatial Searching. In *Proc. ACM SIGMOD Intl. Symp. on the Management of Data*, pages 45–57, 1984.

[KKR90] P. Kanellakis, G. Kuper, and P. Revesz. Constraint query languages. In *Proc. 9th ACM Symp. on Principles of Database Systems*, pages 299–313, Nashville, 1990.

[KKR95] P.C. Kanellakis, G.M. Kuper, and P.Z. Revesz. Constraint query languages. *Journal of Computer and System Sciences*, 51:26–52, 1995.

[KLP00] G. Kuper, L. Libkin, and J. Paredaens. *Constraints Databases*. Springer, 2000.

[Ore86] J. A. Orenstein. Spatial Query Processing in an Object-Oriented Database System. In *Proc. ACM SIGMOD Symp. on the Management of Data*, pages 326–336, 1986.

[PD96] J.M. Patel and D. J. DeWitt. Partition Based Spatial-Merge Join. In *Proc. ACM SIGMOD Symp. on the Management of Data*, pages 259–270, 1996.

[PS85] F. Preparata and M. Shamos. *Computational Geometry: An Introduction*. Springer Verlag, 1985.

[Sch86] A. Schrijver. *Theory of Linear and Integer Programming*. Wiley, 1986.

[SRF87] T. Sellis, N. Roussopoulos, and C. Faloutsos. The R+Tree: A Dynamic Index for Multi-Dimensional Objects. In *Proc. Intl. Conf. on Very Large Data Bases (VLDB)*, pages 507–518, 1987.

[Ull88] J.D. Ullman. *Database and Knowledge Base Systems*. Computer Science Press, 1988.

Appendix

In this appendix we prove the rewriting rules of Fig. 6 and of Fig. 7.

6.1 Intersection Operation: $\cap$

• **windowing**: Intersection between a fixed rectangle *rect* and the geometric information *geo* of R

Lemma 4 $MAP_{\lambda X.[X.geo \cap rect]}\, R \equiv MAP_{\lambda X.[X.geo \cap rect]}\, \sigma_{box \cap rect \neq \emptyset} R$

Proof ($\supseteq$) Obvious. ($\subseteq$) Let $t \in MAP_{\lambda X.[X.geo \cap rect]}\, R$, then there is $u \in R$ such that t is obtained from u and $t.geo = u.geo \cap rect$. By definition $t.geo \neq \emptyset$. Therefore $u.geo \cap rect \neq \emptyset$, and all the projections of $u.geo$ on any set of dimensions are not empty. It follows that $u.box \cap rect \neq \emptyset$ and that $u \in \sigma_{box \cap rect \neq \emptyset} R$. □ The windowing cannot be rewritten using topological relations since the rectangle *rect* is given by the user in the query, and thus doesn't occur in the topological relations.

• Intersection between the geometric parts of two collections R and R'

Lemma 5 $MAP_{\lambda X.[X.geo_R \cap X.geo_{R'}]}\, R \times R' \equiv MAP_{\lambda X.[X.geo_R \cap X.geo_{R'}]}\, \sigma_{box_R \cap box_{R'} \neq \emptyset} R \times R'$

Proof ($\supseteq$) Obvious. ($\subseteq$) Let $t \in MAP_{\lambda X.[X.geo_R \cap X.geo_{R'}]}\, R \times R'$, then there exists $u \in R$ and $u' \in R'$ such that $t.geo = u.geo \cap u'.geo$. By definition $t.geo \neq \emptyset$, so $u.geo \cap u'.geo \neq \emptyset$. Let m be the mbb of $u.geo \cap u'.geo$. Then $m \neq \emptyset$, and all projections of m on any set of dimensions are not empty. Clearly m is included in the intersection of the mbb of u and the mbb of u', and all the projections of m on any set of dimensions are contained in the similar projections of the intersection of the mbb of u and of the mbb of u' on the corresponding dimensions. Thus $u.box \cap u'.box \neq \emptyset$ and u and u' satisfy the selection condition in $\sigma_{box_R \cap box_{R'} \neq \emptyset} R \times R'$. Finally $t \in MAP_{\lambda X.[X.geo_R \cap X.geo_{R'}]}\, \sigma_{box_R \cap box_{R'} \neq \emptyset} R \times R'$. □

Let us consider now the topological relations. To evaluate the query we have to consider that, for each object $o \in R$ and $o' \in R'$ one of the following cases occurs:

1. *disjoint(o,o')*: the two objects are disjoint;
2. *overlap(o,o')*: the result needs to be evaluated on the exact geometric representation;
3. *inside(o,o')*: the geometric intersection is *o.geo* and the result is obtained directly from the filtering;

4. *inside(o',o)*: the geometric intersection is *o'.geo* and the result is obtained with the filtering step.

Finally, using topological relations, the query can be rewritten as:

$$MAP_{\lambda X.[X.geo_R \cap X.geo_{R'}]}\, R \bowtie_{o_id=\$1} overlap \bowtie_{\$2=o_id} R' \cup$$

$$MAP_{\lambda X.[X.geo_R]}\, R \bowtie_{o_id=\$1} inside \bowtie_{\$2=o_id} R' \cup MAP_{\lambda X.[X.geo_{R'}]}\, R' \bowtie_{o_id=\$1} inside \bowtie_{\$2=o_id} R$$

where the first expression in the union considers the case number two (*overlap(o,o')*), the second member refers to case three, and finally last expression considers the case *inside(o',o)*.

6.2 Difference: −

• Difference between the geometric part of R and a fixed rectangle *rect*

Lemma 6

$$MAP_{\lambda X.[X.geo-rect]}\, R \equiv MAP_{\lambda X.[X.geo-rect]}\, \sigma_{box \cap rect \neq \emptyset} R \;\cup\; MAP_{\lambda X.[X.geo]}\, \sigma_{box \cap rect = \emptyset} R$$

Proof ($\supseteq$) Clear if $t \in MAP_{\lambda X.[X.geo-rect]}\, \sigma_{box \cap rect \neq \emptyset} R$. If $t \in MAP_{\lambda X.[X.geo]}\, \sigma_{box \cap rect = \emptyset} R$, there exists $u \in R$ corresponding to t, such that $u.box \cap rect = \emptyset$. Therefore $u.geo - rect = u.geo$, and $t \in MAP_{\lambda X.[X.geo-rect]}\, R$. ($\subseteq$) Let $t \in MAP_{\lambda X.[X.geo-rect]}\, R$. There exists $u \in R$ such that $t.geo = u.geo - rect$. By definition $t.geo \neq \emptyset$, and thus $u.geo - rect \neq \emptyset$. Now either (i) $u.box \cap rect \neq \emptyset$, and $t \in MAP_{\lambda X.[X.geo-rect]}\, \sigma_{box \cap rect \neq \emptyset} R$ or (ii) $u.box \cap rect = \emptyset$, then $u.geo \cap rect = \emptyset$ and $u.geo - rect = u.geo$. Therefore $t \in MAP_{\lambda X.[X.geo]}\, \sigma_{box \cap rect = \emptyset} R$. □

• Difference between the geometric part of two collections R, R'

Lemma 7 $MAP_{\lambda X.[X.geo_R - X.geo_{R'}]}\, R \times R' \equiv$

$$MAP_{\lambda X.[X.geo_R - X.geo_{R'}]}\, \sigma_{box_R \cap box_{R'} \neq \emptyset}(R \times R') \;\cup\; MAP_{\lambda X.[X.geo_R]}\, \sigma_{box_R \cap box_{R'} = \emptyset}(R \times R')$$

Proof ($\supseteq$) Obvious if $t \in MAP_{\lambda X.[X.geo_R - X.geo_{R'}]}\, \sigma_{box_R \cap box_{R'} \neq \emptyset}(R \times R')$. On the other hand, if $t \in MAP_{\lambda X.[X.geo_R]}\, \sigma_{box_R \cap box_{R'} = \emptyset}(R \times R')$, there exists $u \in R$, $u' \in R'$ such that $u.box \cap u'.box = \emptyset$ and $t.geo = u.geo - u'.geo = u.geo$, then $t \in MAP_{\lambda X.[X.geo_R - X.geo_{R'}]}\, R \times R'$. ($\subseteq$) Let $t \in MAP_{\lambda X.[X.geo_R - X.geo_{R'}]}(R \times R')$; there exist $u \in R$, $u' \in R'$ such that $t.geo = u.geo - u'.geo$. By definition $t.geo \neq \emptyset$, and thus $u.geo - u'.geo \neq \emptyset$. One of the two following cases holds: (i) $u.box \cap u'.box \neq \emptyset$, and $t \in MAP_{\lambda X.[X.geo_R - X.geo_{R'}]}\, \sigma_{box_R \cap box_{R'} \neq \emptyset}(R \times R')$. (ii) $u.box \cap u'.box = \emptyset$. In this case, $u.geo \cap u'.geo = \emptyset$ then $u.geo - u'.geo = u.geo$ and finally $t \in MAP_{\lambda X.[X.geo_R]}\, \sigma_{box_R \cap box_{R'} = \emptyset}(R \times R')$. □

Let us consider the topological relations. For each pairs of objects $o \in R$ and $o' \in R'$ we have:

1. *disjoint(o,o')*: the result, o is obtained with no computation;
2. *inside(o,o')*: o is entirely contained in o'. The result is empty and it is obtained with no computation;
3. *inside(o',o)*, *overlap(o,o')*: the result needs to be evaluated on the exact geometric representation;

Then using topological relations, the difference query can be rewritten as

$$MAP_{\lambda X.[X.geo_R]}\, R \bowtie_{o_id=\$1} disjoint \bowtie_{\$2=o_id} R' \cup$$

$$MAP_{\lambda X.[X.geo_R - x.geo_{R'}]}\, R' \bowtie_{o_id=\$1} inside \bowtie_{\$2=o_id} R \cup$$

$$MAP_{\lambda X.[X.geo_R - X.geo_{R'}]}\, R \bowtie_{o_id=\$1} overlap \bowtie_{\$2=o_id} R'$$

6.3 Projection: π

• Projection on one dimension, x_i, of the geometric part *geo* of R.

Lemma 8 If x_i is an attribute of *box*, then $MAP_{\lambda X.[\pi_{x_i} X.geo]}\, R \equiv MAP_{\lambda X.[\pi_{x_i} X.box]}\, R$

Proof ($\supseteq$) Obvious. ($\subseteq$) Let $t \in MAP_{\lambda X.[\pi_{x_i} X.geo]}\, R$, then there is $u \in R$ such that $\pi_{x_i} u.geo = t.geo$. By definition $t.geo \neq \emptyset$. Therefore $\pi_{x_i} u.geo \neq \emptyset$, and all the projections of $u.geo$ on any set of dimensions are not empty. It follows that $\pi_{x_i} u.box \neq \emptyset$. Finally, by definition of *box*, $\pi_{x_i} u.box = \pi_{x_i} u.geo$, and then $t \in MAP_{\lambda X.[\pi_{x_i} X.box]}\, R$. □

• Projection on more than one dimensions, $x_1, .., x_n$.

Lemma 9 If $x_1, .., x_n$ are independent variables[2] in *geo*, and $x_1, .., x_n$ are attributes of *box* then $MAP_{\lambda X.[\pi_{x_1,..,x_n} X.geo]}\, R \equiv MAP_{\lambda X.[\pi_{x_1,..,x_n} X.box]}\, R$

Proof ($\supseteq$) Let $t \in MAP_{\lambda X.[\pi_{x_1,..,x_n} X.box]}\, R$. There exist $u \in R$ such that $t = \pi_{x_1,..,x_n} u.box$. Therefore $x_1, .., x_n$ are independent variables, and they are also attributes of *box*. By definition of *box*, $\pi_{x_1,..,x_n} u.box = \pi_{x_1,..,x_n} u.geo$, and then $t \in MAP_{\lambda X.[\pi_{x_1,..,x_n} X.geo]}\, R$. ($\subseteq$) Let $t \in MAP_{\lambda X.[\pi_{x_1,..,x_n} X.geo]}\, R$, then there is $u \in R$ such that $\pi_{x_1,..,x_n} u.geo = t.geo$. By definition $t.geo \neq \emptyset$. Therefore $\pi_{x_1,..,x_n} u.geo \neq \emptyset$, and all the projections of $u.geo$ on any set of dimensions are not empty. It follows that $u.box \neq \emptyset$, $\pi_{x_1,..,x_n} u.box \neq \emptyset$. Therefore $x_1, .., x_n$ are attributes of *box*. By definition of *box*, and for the independence of variables $\pi_{x_1,..,x_n} u.box = \pi_{x_1,..,x_n} u.geo$, and then $t \in MAP_{\lambda X.[\pi_{x_1,..,x_n} X.geo]}\, R$. □

6.4 Selection: σ

• Select the geometric objects that contain a given point $P(a_1, .., a_k)$

Lemma 10 $MAP_{\lambda X.[\sigma_{x_1=a_1 \wedge .. \wedge x_k=a_k} X.geo]}\, R \equiv MAP_{\lambda X.[\sigma_{x_1=a_1 \wedge .. \wedge x_k=a_k} X.geo]}\, \sigma_{P \in box} R$, where the selection condition $P \in box$ holds if the coordinates of P satisfy the constraints of *box*.

Proof($\supseteq$) Obvious. ($\subseteq$) Let $t \in MAP_{\lambda X.[\sigma_{x_1=a_1 \wedge .. x_k=a_k} X.geo]}\, R$, then there exists $u \in R$ such that $P \in u.geo$ and $t.geo = u.geo$. By definition of *box*, $P \in u.box$, then $u \in \sigma_{P \in box} R$, and finally $t \in MAP_{\lambda X.[\sigma_{x_1=a_1 \wedge .. \wedge x_k=a_k} X.geo]}\, \sigma_{P \in box} R$. □

• Select the geometric part *geo* of R, that satisfy the constraint F.

Lemma 11 $MAP_{\lambda X.[\sigma_F X.geo]}\, R \equiv MAP_{\lambda X.[\sigma_F X.geo_A]}(\underbrace{MAP_{\lambda X.[X.geo, \sigma_F X.box]}\, R}_{A})$

Proof($\supseteq$) Obvious. ($\subseteq$) Let $t \in MAP_{\lambda X.[\sigma_F X.geo]}\, R$. There exists $u \in R$ such that $t.geo = \sigma_F u.geo$. By definition $t.geo \neq \emptyset$, and thus $u.geo \neq \emptyset$. Then all the projections of $u.geo$ on any set of variables are not empty. It follows that $u.box \neq \emptyset$. By definition of *box*, $u.box$ satisfy the selection condition F, then $t \in MAP_{\lambda X.[\sigma_F X.geo]}(MAP_{\lambda X.[X.geo, \sigma_F X.box]}\, R)$. □

6.5 Join: $\bowtie$

• Join between the geometric attributes of two collections R and R'.
In similar way as in Lemma 5 it is possible to prove the following

Lemma 12 $MAP_{\lambda X.[X.geo_R \bowtie X.geo_{R'}]}\, R \times R' \equiv MAP_{\lambda X.[X.geo_R \bowtie X.geo_{R'}]}\, \sigma_{box_R \cap box_{R'} \neq \emptyset} R \times R'$

Using the topological relation *overlap*:

$$MAP_{\lambda X.[X.geo_R \bowtie X.geo_{R'}]}\, R \times R' \equiv MAP_{\lambda X.[X.geo_R \bowtie X.geo_{R'}]}\, R \bowtie_{o_id=\$1} overlap \bowtie_{\$2=o_id} R'$$

[2] Let $\varphi(x_1, \ldots, x_d)$ be a formula in $\mathcal{L}$ with d distinct free variables. Two distinct variables are *independent* is they never occur in the same atom [CGK96].