Spatial Database Querying with Logic Languages

Jean-Pierre Cheiney & Vincent Oria
Ecole Nationale Supérieure des Télécommunications
75013 Paris - France
{cheiney, oria}@info.enst.fr

Abstract
Several different data structures, generally grouped in Raster and Vectors representation models, are used to store images and all kinds of spatial data. One solution, for a spatial data manipulation language to be independent of the storage model, is to base the language on the spatial relations of objects (i.e. the positions of objects relative to each other). This paper proposes a logic intermediate language allowing a declarative querying. We show that this language can be easily computed by a procedural execution using a small set of operators. This language enables expression of direction and topological spatial relations and ensures a physical data independence and the processing can benefit from spatial access methods. The language is composed of two complementary sub-languages. The first one uses an object approximate and results in the selection of a set of candidate objects, which are investigated more in detail using the second one.

1 Introduction
Most database query languages for spatial data are based on specific data structures used for image storage. These structures correspond to two representation models: raster [22, 11] in which the image is considered as a set of points, and vector [17] in which the objects are stored as a set of vectors. The choice of a specific data structure depends on the forecasted manipulations. Such languages are linked with a particular application domain and various uses demand different representation of the images [16]. In this paper we focus only on the querying of spatial data with a minimum model (a spatial object is a set of points).

An interesting approach that takes into account both of the representation models, is to build the data manipulation language using spatial relations existing between objects (elements of the images). In spatial relations, we assume relative positions of objects with respect to each other. The spatial relations allow the description of constraints on the objects, for searching or updating particular objects. There are two kinds of spatial relations: direction relations and topological relations. Direction relations are relations such as left, right, above, below [7, 5] or east, west, north and south in geographic applications [20, 8, 12]. Topological relations [10, 9] involve the boundary, interior and exterior of objects such as disjoint, meet, equal, covers, covered by, inside, contains and overlap. Most of previous works on spatial relations studied either direction relations or topological relations between objects. Otherwise, the operators are built for a specific image representation and use specific data structures. Works on direction relations have focused on how to combine knowledge about directions. The 8 spatial topological relations between the objects are represented by Egenhofer using a 3x3 matrix called 9-intersections [10]. For two regions (objects with surface) A and B, Egenhofer compares the closure ($\delta A$, $\delta B$), the interior ($A_0$, $B_0$) and the exterior ($A_-$, $B_-$). The 9 possible combinations constitute the elements of the 9-intersection matrix. This approach is interesting but is linked to the storage data structures of the objects (e.g. a raster storage implies 8 other relations) [9] and cannot express direction relations. Another approach is to project the objects on the axes and to study their distribution on the different axes. Basic relations in a one-dimensional (1D) space (each of the axes) are defined and their combination leads to define complex direction relations in a nD space. This approach was used in [13] to support topological and direction spatial relations, but stays limited to the objects having rectangular shapes. We have followed the same approach [6, 16], considering the relations between object projections as filter relations and proposing another solution for the refinement which overcome the limits of Guesgen's approach.

We consider two classes of spatial queries: overlap queries and disjunction queries. The first class groups queries involving proximity, neighbourhood and other topological spatial relations except the disjoint relation.
which can be expressed by the negation of the others or by a query of the second class. The general form of a query of this class is "retrieve the pairs \((d \in D, q \in Q)\) such that \(d\) and \(q\) overlap", given two sets \((D\) and \(Q)\) of objects. The second class of queries relies on axes and contains all the queries such that the involved objects do not intersect. They include direction relations predicates like "north of", "south of", "east of", "west of", "left of", etc. The evaluation of these queries is often processed in two or more steps [8, 3, 4]:

- a filter step which allows the selection of a set of candidate objects using a crude approximation of the objects.
- a refinement step which processes the objects selected in the first step and produces the result.

The evaluation of these queries is often processed in two or more steps [8, 3, 4]:

- a filter step which allows the selection of a set of candidate objects using a crude approximation of the objects.
- a refinement step which processes the objects selected in the first step and produces the result.

The main contributions of this paper are: (1) to propose an intermediate non-procedural language for expressing the queries (descriptive capability), (2) to show that the evaluation of the ProLo-PaLo queries can be made by few operators (computing capability) applying on data, (3) to ensure a physical data independence by building functions \(\Pi\) and \(\lambda\) on different data representation formats (raster or vectors).

The remainder of this paper is organized as follows: Section 2 introduces the ProLo language, the PaLo language is presented in Section 3. The 4th Section shows how to use ProLo and PaLo languages in the spatial database context. We suggest possible extensions to take into account objects with holes and non-connected objects in Section 5.

2 The Projection Logic ProLo

The Projection Logic allows the use of predefined boolean operators on segments obtained from projection on the axes. Thus, the study of spatial relations between objects in a \(nD\) space is transformed into \(n\) studies for each elementary 1D domain. The use of objects projection is well known in spatial information searching. It is used, in particular, in the 2D string index structures [5]. The example of Figure 2 illustrates an intersection of the object \(O_1\) and \(O_2\) on the axis 1 and a precedence relation between \(O_2\) and \(O_1\) on the axis 2.

The intersection and precedence concepts are directly used in ProLo Language with the introduction of the "intersect" and "before" operators. These two operators are used to define atomic formulae which support the functions \(\Pi\) (projection of objects on the axes, a requirement of the ProLo Language) and \(\lambda\) (intersection of a straight line with an object, a requirement of the PaLo language) as shown in Figure 1.
allow the construction of more complex logical formulae by combination with logic constructors. First we will explain the object’s projection, then operators and language and lastly, how filtering is done.

2.1 The Projection

Before defining the projection, let us give a formal definition of objects.

Let \( D = D_1 \times \ldots \times D_n \) be a space of dimension \( n \).

We call \( D_i \), the PowerSet of \( D_i \). In the same way, we note by \( D_i \) the PowerSet of \( D_i \).

**Definition 1:** a point is an atomic element of \( D \).

**Definition 2:** \( O \) is an object of \( D \) if \( O \) is an element of \( D \).

**Definition 3:** Connected object: an object \( O \) is connected, if for all pairs of points \( P \) and \( Q \) from \( O \), a connected path exists between them; i.e. a sequence of points exists beginning at \( P \) and ending at \( Q \) and all these points belong to \( O \).

An object can be connected or not, with holes or without holes. In particular we use the following objects in a 2D space:

- a point which is used to associate a position in space to an object. For example, a town can be represented by a point;
- a line is a sequence of linear connected segments;
- a polygon is represented by its borders. Its position is given by a closed chain of points. We can distinguish simple polygons and complex polygons, with holes or without holes. In general, only simple polygons are considered in spatial database and viewed as areas. Initially, we will consider only simple objects i.e. without holes. The other cases will be added afterwards.

Let \( O \) be an object of \( D \) and \( z = (x_1, \ldots, x_n) \) be a point of \( D \). We call projection of \( O \) on the sub-space \( D_i \), \( i \in \{1 \ldots n\} \) the function noted by \( \Pi_i \):

\[
\Pi_i : D \rightarrow D_i \quad O \rightarrow \Pi_i(O) = \{ y \in D_i / \exists z \in O \wedge x_i = y \}
\]

More generally, we note \( \Pi \), the function which gives the set of the projections:

\[
\Pi : D \rightarrow D_1 \times \ldots \times D_n \quad O \rightarrow (\Pi_1(O), \ldots, \Pi_n(O))
\]

We call MBB (Minimum Bounding Box) - MBR (Minimum Bounding Rectangle) in a 2D space, the function that makes an object correspond to the minimal structure which includes it. The MBB\((O)\) of an object \( O \) is an object characterized by the cartesian product of the different projections \( \Pi_i(O) \), with \( i \) from 1 to \( n \):

\[
MBB : D \rightarrow D \quad O \rightarrow MBB(O) = \Pi_1(O) \times \ldots \times \Pi_n(O)
\]

It is clear that \( \forall O \in D, O \subseteq MBB(O) \)

2.2 The ProLo Spatial Operators

The projection on the axes gives a rough representation of objects. Precise definitions of the adjacency, the inclusion and the overlap are not necessary. We group these three operations to only one which checks the intersection. So, the ProLo language has only two operators by axis. These operators, before and intersect, are defined by the following expressions for an axis \( i \):

\[
i \text{ before}(<i) : \quad (O_1 <i O_2) \iff \forall x \in \Pi_i(O_1), \forall y \in \Pi_i(O_2) x < y
\]

\[
i \text{ intersect}(\cap_i) : \quad (O_1 \cap_i O_2) \iff \exists z, z \in \Pi_i(O_1) \wedge z \in \Pi_i(O_2)
\]

The operators \( \cap_i, <_i \) of the ProLo language are boolean operators. \((O_1 \cap_i O_2)\) is true iff \(\Pi_i(O_1)\) intersects \(\Pi_i(O_2)\). In the same way, \((O_1 <_i O_2)\) is true iff \(O_1\) is "before" \(O_2\) on the axis \(i\). The spatial relations between the objects \(O_1\) and \(O_2\) of Figure 2 is given in the ProLo language by the expression \((O_1 \cap_i O_2) \wedge (O_2 <_i O_1)\). The "before" operator can be used to define spatial direction relations like "east of", "south of", "above" and "below".

**Properties:** Let \( O_1 \) and \( O_2 \) be two objects, we can assert, for connected objects, that:

\[
\neg(O_1 \cap_i O_2) \iff (O_1 <_i O_2) \vee (O_2 <_i O_1)
\]

\[
\neg(O_1 <_i O_2) \iff (O_1 \cap_i O_2) \vee (O_2 <_i O_1)
\]

With these properties, we can define only the before operator for the ProLo language.

**Note:** The definition of the before operator can be too strong for some applications. Anyhow, it is possible to define a function \( \text{Centre} \) that gives the centre of the MBB of an object that will be used as follows for a more flexible before operator definition:

\[
(O_1 <_i O_2) \iff \Pi_i(\text{Centre}(O_1)) < \Pi_i(\text{Centre}(O_2))
\]

For the remainder of this paper we will concentrate on the first definition of the before operation.
2.3 ProLo Formulae Definition

- **terms**: a constant object and an object variable are terms.
- **atomic formula**: if $O_1$ and $O_2$ are terms, then $O_1 \theta O_2$, where $\theta \in \{\prec_1, \ldots, \prec_n, \cap_1, \ldots, \cap_n\}$, is an atomic formula.
- **logic connectives**: if $f$ and $g$ are formulae then $f \wedge g$ is a formula; $\neg f$ is a formula.

The other logic connectives are introduced as abbreviations. The connective propositions $f \vee g$, $f \Rightarrow g$, $f \iff g$ abbreviate respectively $\neg (\neg f \wedge \neg g)$, $\neg f \vee g$, $(f \wedge g) \vee (g \wedge f)$. The boolean constant true is the abbreviation of $f \vee \neg f$ and false is the abbreviation of $\neg f \vee f$.

2.4 ProLo: A Filter Language

To show the filter characteristic of the ProLo language, we have to demonstrate the following result: for all spatial relations of disjunction or overlap true for two objects, there exists a ProLo Language expression, involving the same spatial relations true for the MBB of the objects and for the same relation.

**Proposition 1**: Let $F$ be a formula expressing an overlap or a disjunction spatial relation between two objects $O_1$ and $O_2$, then there exists a formula $F_p$ of the ProLo language such as $F \iff F_p$.

**Proof**: $F$ has the form $F(O_1, O_2)$ and $F_p$ has the form $F_p(O_1, O_2)$. $F_p$ can be put under the form $F(MBB(O_1), MBB(O_2))$ Since $O_1 \subseteq MBB(O_1)$ and $O_2 \subseteq MBB(O_2)$, we have $\neg F_p(O_1, O_2) \iff \neg F(O_1, O_2)$.

The using of approximations with the MBB is a classical method [3] in spatial databases as it is simple and corresponds to existing access methods based on R-trees [15] (or one of its variants) that can be used for an efficient implementation of the ProLo language. The Prolo language, working on the MBB of the objects, gives the cases where objects do not intersect. The result of a ProLo query, in the case of a MBB intersection demands a finer scanning with PaLo Language.

3 PaLo: A Refinement Language

Whilst ProLo operates from the axes, PaLo uses straight lines parallel to the axes called paths and study the relations between parts of objects which intersect these lines. The path is concretized by fixing as a constant the values of $n - 1$ components of a $nD$ space.

The domain for which there is no constant is called active domain. In the example of Figure 3, $\lambda_1$ and $\lambda_2$ are paths with the domain of axis 1 as the active domain. If we consider the intersection of the path $\lambda_1$ with the respective objects $O_1$, $O_2$ and $O_3$, we obtain the segments $\lambda_1(O_1)$, $\lambda_1(O_2)$ and $\lambda_1(O_3)$ which we can compare. For example, $\lambda_1(O_1)$ and $\lambda_1(O_2)$ are adjacent and $\lambda_1(O_1)$ is before $\lambda_1(O_2)$. We will now define formally the path concept, the operators of the PaLo language and the language itself. A discussion on the choice of paths ends this section.
Note that $\lambda(O) \subseteq O$ and $\lambda(O) \in D_\lambda$.

The parts of an object within a path $\lambda$ satisfies Propositions 2 and 3 which express respectively that the projection of a part of an object on an axis is contained in the projection of the object on this axis and that the union of all the parts of the object composes the object in a discrete working space.

**Proposition 2**: Let $D_i = DA(\lambda), \forall \vec{c} \in D \setminus D_i \exists \lambda s.t. (\text{coeff}(\lambda) = \vec{c} \wedge (\lambda(O) \neq \emptyset)) \Rightarrow (\Pi_i(\lambda(O)) \subseteq \Pi_i(O))$.

**Proof**: $\vec{c} \in \lambda(O) \Rightarrow \exists x \in D_i, \sigma^c_i(x) = \vec{c}$ so $x \in \Pi_i(O)$.

**Proposition 3**: For a discrete working space $D$, $\forall D_i = DA(\lambda), u_{\text{coeff}}(\lambda)(O) = O$.

**Proof**: Let $D_i$ be an active domain, let $O$ be an object, and let $\vec{c} = (x_1, \ldots, x_n)$ be a point.

Either $\vec{c} \in O$, or $\vec{c} \notin O$.

$\vec{c} \notin O$ : nothing.

$\vec{c} \in O \Rightarrow \exists \vec{c} \in D \setminus D_i, \exists \lambda \text{coeff}(\lambda) = \vec{c}, \exists z \in D_i \vec{c} \in \lambda(O) \wedge \sigma^c_i(x) = \vec{c}$.

### 3.2 PaLo Spatial Operators

Let $\lambda : \vec{y} = \vec{x}$, be a path with an active domain $D_i$ and let $\vec{x} = (x_1, \ldots, x_n)$ and $\vec{y} = (y_1, \ldots, y_m)$ be variables of $D$, we declare for $\lambda$ the following operators:

- $\lambda_{\text{before}}$:
  $$\lambda_{\text{before}} : (O_1 \prec_\lambda O_2) \iff \forall \vec{x} \in \lambda(O_1), \forall \vec{y} \in \lambda(O_2) x_i < y_i$$

- $\lambda_{\text{join}}$:
  $$\lambda_{\text{join}} : (O_1 \sqsubseteq_\lambda O_2) \iff \forall \vec{x} \in \lambda(O_1), \forall \vec{y} \in \lambda(O_2) x_i \leq y_i$$

- $\lambda_{\text{overlap}}$:
  $$\lambda_{\text{overlap}} : (O_1 \sqcap_\lambda O_2) \iff \forall \vec{x} \in \lambda(O_1), \forall \vec{y} \in \lambda(O_2) x_i < y_i$$

- $\lambda_{\text{in}}$:
  $$\lambda_{\text{in}} : (O_1 \sqsubseteq_\lambda O_2) \iff \forall \vec{x} \in \lambda(O_1), \forall \vec{y} \in \lambda(O_2) x_i \leq y_i$$

In the example of Figure 4, according to the path $\lambda_1$, the expression of spatial relations is given by $(O_1 \prec_\lambda, O_2) \cap (O_1 \prec_\lambda_1, O_2) \cap (O_2 \sqsubseteq_\lambda O_3)$ and according to the path $\lambda_2$, we have $(O_1 \sqsubseteq_\lambda_2, O_4)$.

### 3.3 PaLo Language Definition

The PaLo language is a two sorted logic, one sort is called object and the other one is called path.

- **terms of type object**: A constant object, a variable on an object are terms of type object.

- **terms of type path**: A constant path, a variable on a path are terms of type path.

- **atomic formula**: if $O_1$ and $O_2$ are terms of sort object and $\lambda$, a path of coefficient $\vec{c}$ with an active domain $D_i$, then $O_1 \theta O_2$ where $\theta \in \{\ll_\lambda, \preceq_\lambda, \sqsubseteq_\lambda, \sqcap_\lambda, \sqcup_\lambda\}$ is an atomic formula.

- **quantifiers** (are used only for the paths): if $\lambda$ is a free variable of sort path and $f_\lambda$ a formula defined on the path $\lambda$ then $\exists \lambda f_\lambda$ is a formula, $\forall \lambda f_\lambda$ is a formula.

- **logic operators**: if $f$ and $g$ are formulae then $f \land g$ is a formula, $\neg f$ is a formula.

The other logic operators are appended as abbreviations. A PaLo formula can be true, false or undefined. A PaLo formula within a path $\lambda$ is undefined if the path does not cut all the objects involved in the formula. In the case of Figure 3, the expression $(O_4 \sqsubseteq_\lambda O_2)$ is undefined. The performances of PaLo language depends on a good choice of the paths.

### 3.4 Optimal Choice of Paths

Let $D_i$ be an active domain. It is too expensive to examine all the possible coefficients. So, the system has to decide one precise set of values where it will be able to find correct path coefficients.

Let $C(O)$ be the set of coefficients $\vec{c} \in D \setminus D_i$ so that the path $\lambda : \vec{y} = \vec{x}$ cuts the object $O$ (i.e. an element $x \in D_i$ exists such that $(\vec{c}, x) \in O$).

**Proposition 4**: $C(O) \subseteq \Pi_1(O) \times \ldots \times \Pi_{j-1}(O) \times \Pi_{j+1}(O) \times \ldots \times \Pi_n(O)$.

**Proof**: the proof is given by the fact that $\Pi_{p_j}(O) \subseteq \Pi_1(O) \times \ldots \times \Pi_{j-1}(O) \times \Pi_{j+1}(O) \times \ldots \times \Pi_n(O)$.

Remark: for a 2D space and for an active domain $D_i$, we have exactly $C(O) = \Pi_j(O)$ with $i \neq j$.

Proposition 4 allows the space of the quantifiers of paths to be limited to a set of given paths. The statement allows one and only one active domain for a plan.
Proposition 5: All algorithms checking the intersection of the objects $O_1$ and $O_2$ are implementations of $\exists \lambda f_3(O_1, O_2)$ where $f_3(O_1, O_2)$ is a formula in PaLo expressing the intersection of the objects $O_1$ and $O_2$.

Proof: $\exists \tilde{y} \in O_1 \land \tilde{z} \in O_2 \iff \exists \lambda O_1 \theta O_2$, where $\theta \in \{\subseteq, \supseteq, \cap, \cup\}$ and $\text{coeff}(\lambda) = \Pi_{\phi \in \lambda}(\tilde{y}) \land DA(\lambda) = D_1$.

Proposition 5 allows the PaLo language to be distinguished from all the possible implementations, using known geometrical algorithms [21] for efficient implementation of the operators.

4 ProLo-PaLo: A Query Language for Spatial Data

A simple method to obtain safe queries is to restrict the variables, i.e. to limit the values to those ones which are in the queries or to those ones which are input values [1]. In our case, we are going to restrict, with an implicit formula, the variables of sort object to the objects of a limited set $G$ and to these which are in the query and the variables of sort path to a predefined set of paths.

A ProLo query: A ProLo query is an expression $q_p(G) = \langle O \in G \land F_p(O) \rangle$ where $G$ is a limited set of objects and $F_p(O)$ a formula of the ProLo language, with $O$ as an object variable. A ProLo query $q_p(G)$ is defined as follows: $q_p(G) = \{ O \in G \land F_p(O) \}$

A PaLo query: A PaLo query is an expression $q_c(G) = \langle O \in G \land F_c(O) \rangle$ where $G$ is a limited set of objects and $F_c(O)$ a formula of PaLo with $O$ as an object variable. A PaLo query $q_c(G)$ is defined as follows: $q_c(G) = \{ O \in G \land F_c(O) \}$

4.1 ProLo-PaLo: A Language for Overlapping and Disjunction Queries

Combining these two languages leads to efficient answers of spatial queries. The PaLo language is designed for spatial direction queries and for filter queries that have to select a set of candidate objects in the case of overlapping queries. When refining a set result of an overlapping ProLo query, we are sure that there exists, at least, a path cutting the objects involved in the query for a PaLo overlapping query.

Proposition 6: All overlapping and disjunction spatial queries are totally computed by ProLo and PaLo.

Proof: Let $\text{ProLo}(O_1, O_2)$ be a formula of ProLo defining a spatial relation between the objects $O_1$ and $O_2$. Either $\text{ProLo}(O_1, O_2)$ shows that the objects MBB intersect, or that they do not.

- $\text{ProLo}(O_1, O_2)$ shows that the MBB of the objects do not intersect, then the objects do not either;
- $\text{ProLo}(O_1, O_2)$ shows that the MBB of the objects intersect then,

if, for any active domain, there exists a path $\lambda$ so that $\lambda(O_1)$ and $\lambda(O_2)$ intersect, then the objects intersect, otherwise they do not. The opposites are also true.
the Palo interrogation is the result \( q_p(G) \) of the Prolo interrogation.

\[ q_p(G) = \{ O \in G \mid \forall \lambda \text{ coeff(}\lambda\text{)} \in C(O) \cap C(O_\lambda), O \supset \lambda \}
\]

\( C(O) \) is the set of the path coefficients which cut the object \( O \). This leads to the result set \( \{O_2, O_4, O_5\} \)

\[ \alpha \text{ between } \] and \[ \beta \text{ between } \]

\( \alpha = (x_1, \ldots, x_n) \) variables of \( D \), \( (x_1, \ldots, x_n) \) and

\( \beta = (y_1, \ldots, y_n) \) variables of \( D \), \( (y_1, \ldots, y_n) \) and

\( \lambda \text{ between } \)

\[ \exists = (x_1, \ldots, x_n) \text{ variables of } D, (O_1, O_2, O_3) \text{ iff:}
\]

\[ (\exists \gamma \in \lambda(O_2), \exists \eta \in \lambda(O_3), \forall \gamma \in \lambda(O_1), y_i < x_i < z_i)
\]

\[ \wedge (\forall \gamma \in \lambda(O_2), \forall \eta \in \lambda(O_3), z_i \neq y_i)
\]

The extension to non-connected objects demands to be taken into account in the ProLo language, for each axis, an operator similar to \( \lambda \text{ between } \) that we call \( \iota \text{ between } \) noted \( O_i \) with a similar definition.

6 Conclusion

The Projection Logic leads to the study of relations between objects of \( n \) dimension space to \( n \) studies for each elementary domain. To get here, we first project the objects on the axes. This allows us to work on approximations constituted by the Minimum Bounding Boxes (MBB). We so define a filter language ProLo. On the contrary, the Palo language uses straight lines called paths. On these paths, the intersections with the parts of objects, are studied. The path is materialised by fixing as constants the values of \( n - 1 \) components of a space of dimension \( n \) and by combining with the values of domain of the chosen axis (active domain). The Palo language is useful when a path cutting the objects exists. This includes the case when there is an intersection between objects MBB, in this case it is not possible to conclude with the ProLo language. Hence, we have shown that the composition of these two languages allows the computation of all overlapping or disjunction spatial queries, with respect to a filter step and a refining step. ProLo-Palo is more expressive than other languages proposed for topological spatial reasoning and direction spatial reasoning as it can express both.

Moreover, the idea of basing these languages only on the spatial relations between objects excludes the choice of a data storage model and leads us to see these languages in a spatial data model. The ProLo-Palo language can be easily implemented using spatial indexes of R-trees family and that any algorithm that checks object intersection is an implementation of Palo. Anyway, the manipulated pieces of data are intervals both in Palo and in ProLo and this allows the same type of processing for the two languages us-
ing object decomposition into trapezoids [2] or other computational geometry algorithms [21].

References


