Identifying Topological Predicates for Vague Spatial Objects

Alejandro Pauly & Markus Schneider*
University of Florida, Department of Computer & Information Science & Engineering
Gainesville, FL 32611, USA
{apauly, mschneid}@cise.ufl.edu

ABSTRACT

Many geographical applications deal with spatial objects that cannot be adequately described by determinate, crisp concepts because of their intrinsically indeterminate and vague nature. GIS and spatial database systems are currently unable to handle this kind of data. Based on recent work on *vague spatial data types*, which are part of a formal data model called *VASA* (*Vague Spatial Algebra*) and which leverage exact models of crisp spatial data types, this paper introduces a general mechanism for identifying topological predicates for vague spatial objects by means of topological predicates for crisp spatial objects. We illustrate this mechanism by deducing these predicates for vague points.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Spatial Databases and GIS

General Terms

Design

Keywords

Vague spatial data type, VASA, three-valued logic, cancellation rule, clustering rule, query language

1. INTRODUCTION

Current spatial data models and their implementations in GIS and spatial database systems are based on the illusory premise that all spatial objects can be adequately represented as exclusively crisp and exactly determined entities. That is, they implicitly assume that the positions of points, the locations and routes of lines, and the extent and hence the boundary of regions are precisely determined and universally recognized. Examples are man-made spatial objects (e.g., monuments, highways, buildings) and immaterial spatial objects (e.g., countries, districts, land parcels with their political, administrative, and cadastral boundaries). We denote these entities as *crisp spatial objects*.

On the other hand, for many geographic applications this premise is unfounded and inappropriate since the feature of *spatial vagueness* is inherent to their data. Positions of points turn out to be not exactly known, the locations and routes of lines are unclear, and regions do not have sharp boundaries, or their boundaries cannot be sharply determined. Examples are social or natural phenomena like terrorists' refuges and escape routes, population density, unemployment rate, soil quality, vegetation, and oil fields. We denote these entities as *vague spatial objects*.

Whereas topological predicates have been largely investigated for crisp spatial objects, this is not the case for vague spatial objects. The goal of this paper is to develop a general mechanism for identifying topological predicates for vague spatial objects instead of several individual procedures for each single type combination. These *vague topological predicates* are part of a formal data model called *VASA* (*Vague Spatial Algebra*) and are based on recent work on so-called *vague spatial data types* [5] introducing *vague points*, *vague lines*, and *vague regions*. This paper shows how vague topological predicates can be deduced from well explored crisp topological predicates.

Section 2 presents related work. Section 3 introduces vague spatial data types. Our general mechanism for identifying vague topological predicates is explained in Section 4. In Section 5 we demonstrate this mechanism for predicates between vague points. Section 6 shows how these predicates can be used for querying. Finally, Section 7 draws some conclusions and addresses future work.

2. RELATED WORK

Spatial vagueness is an intrinsic feature of a spatial object for which we cannot be sure whether certain components belong to the object or not. Models based on *fuzzy sets*, *rough sets*, and *exact spatial objects* have been proposed so far as general design methods for them. A discussion of their differences can be found in [7]. Vague spatial data types [5] belong to the latter models.

The basis of the latter model category are crisp spatial data types like *point*, *line*, and *region* (see [6] for a survey). Much research on spatial databases has been devoted to topological predicates (like *overlap*, *disjoint*) on crisp spa-

*This work was partially supported by the National Science Foundation under grant number NSF-CAREER-IIS-0347574.

SAC'05 March 13-17, 2005, Santa Fe, New Mexico, USA

tial data types. Our definition of vague topological predicates rests on topological predicates that are defined on crisp *complex* spatial objects and not on *simple* spatial objects as in the other approaches. Complexity means here that *point* objects are finite collections of points, *line* objects are finite collections of disjoint curves, and *region* objects are finite collection of disjoint faces possibly with holes.

Topological predicates for simplified vague regions have already been studied in [3, 4]. These approaches suffer from two drawbacks. First, the crisp regions used are only simple (i.e., single-component, hole-free) regions. Second, the vague regions defined are too restrictive in the sense that they are regions with "broad boundaries". That is, one crisp simple region, whose area definitely belongs to the vague region, is located inside another larger crisp simple region. Their geometric difference is considered to be the broad, vague boundary.

3. VAGUE SPATIAL DATA TYPES

For our definition of vague spatial data types, we consider a homeland security scenario as an illustrating example. Secret services (should) have knowledge of the whereabouts of terrorists. Some of a terrorist's refuges are precisely known, some are not and only conjectures. We can model these locations as a *vague point* object where the precisely known locations are given by a *kernel point* object and the assumed locations are described by a *conjecture point* object. Secret services are also interested in the routes a terrorist takes to move from one refuge to another. These routes can be modeled as *vague line* objects. Some routes, summarized in a *kernel line* object, have been definitely identified. Other routes, specified as a *conjecture line* object, can only be assumed to be taken by a terrorist. Knowledge about areas of terroristic activities is also important for secret services. From some areas, described as a *kernel region* object, it is well known that a terrorist operates in them. From other areas, given by a *conjecture region* object, we can only assume that they are a target of terroristic activity. Figure 1 gives some illustrations. Grey shaded areas, straight lines, and grey points indicate kernel parts; areas with white interiors, dashed lines, and white points refer to conjecture parts.

For the definition of vague points, vague lines, and vague regions we leverage the well known data types *point* for crisp points, *line* for crisp lines, and *region* for crisp regions [6]. These types are closed under the geometric set operations $\oplus$ (*union*), $\otimes$ (*intersection*), $\ominus$ (*difference*), and $\sim$(*complement*). The use of an exact model for constructing vague spatial data types leads to the benefit that existing definitions, techniques, data structures and algorithms need not be redeveloped but can simply be used or in the worst case slightly modified or extended as necessary.

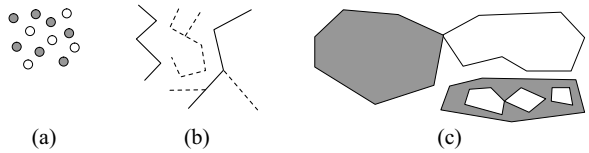

Figure 1: Examples of a *vague point* object (a), a *vague line* object (b), and a *vague region* object (c).

A vague spatial object is described by a pair of two *disjoint* or *meeting* crisp complex spatial objects. Hence, a generic definition applies to all vague spatial data types. That is, the extension of a crisp spatial data type to a corresponding vague type is given by a type constructor v as follows:

$$v(\alpha) \quad = \quad \alpha \times \alpha \qquad\qquad \forall\, \alpha \in \{point, line, region\}$$

with the additional constraint that

$$\forall\, \alpha \in \{point, line, region\}\ \forall\, w = (w_k, w_c) \in v(\alpha) : \\ disjoint(w_k, w_c) \ \vee \ meet(w_k, w_c)$$

This means that for $\alpha = point$ we obtain $v(point) = point \times point$, which we also name *vpoint*. Accordingly, the data types *vline* and *vregion* are defined. For a vague spatial object $w = (w_k, w_c) \in v(\alpha)$, the first crisp spatial object w_k, called the *kernel part*, describes the component that definitely and always belongs to the vague object. The second crisp spatial object w_c, called the *conjecture part*, describes the vague component of w, for which we cannot say with any certainty whether it or subparts of it belong to the vague object or not. *Maybe* the conjecture part or subparts of it belong to the vague object, *maybe* this is not the case.

4. THE GENERAL MECHANISM

Topological predicates provide information about the relative position of spatial objects towards each other. Since vague spatial objects include a conjecture part, our well known binary logic with *true* and *false* as its only values turns out to be inappropriate. The result type of *vague topological predicates* is thus a new vague data type named $vbool = \{true, false, maybe\}$ $(= \{t, m, f\})$; it is the basis of a *three-valued logic*. The definition of the *vague logical connectors* **and**, **or**, and **not** reflecting the influence of the *maybe* value on logical operations is given in Table 1.

and	t	m	f
t	t	m	f
m	m	m	f
f	f	f	f

or	t	m	f
t	t	t	t
m	t	m	m
f	t	m	f

not	t	m	f
	f	m	t

Table 1: Vague logical operators.

Our first goal is to find a mechanism that in a systematic, consistent, and correct way identifies the topological predicates for all combinations of vague spatial data types. This avoids designing a number of different derivation methods for different type combinations. Our second goal is to make use of well known definitions of topological predicates on crisp spatial objects. Our third goal is to benefit from great advantages with respect to implementation. Vague topological predicates can later make use of preexisting implementations of crisp topological predicates and be implemented on top of them as executable specifications.

Our approach illustrated in Figure 2 aims at a characterization of topological predicates on vague spatial objects by means of conjunctions of topological predicates on complex crisp spatial objects. For $\alpha, \beta \in \{point, line, region\}$ let $T_{\alpha,\beta}$ be the type-combination specific set of topological predicates. The work in [1], e.g., identifies five (33) topological predicates between two complex points (regions). Given a vague spatial object $A = (A_k, A_c)$, we get access to the crisp spatial objects A_k, A_c, and $A_k \oplus A_c$. Correspondingly,

a vague spatial object $B = (B_k, B_c)$ enables the access to the crisp spatial objects B_k, B_c, and $B_k \oplus B_c$. The idea now is to determine the uniquely defined topological predicate for the nine pairs (A_k, B_k), (A_k, B_c), $(A_k, B_k \oplus B_c)$, (A_c, B_k), (A_c, B_c), $(A_c, B_k \oplus B_c)$, $(A_k \oplus A_c, B_k)$, $(A_k \oplus A_c, B_c)$, and $(A_k \oplus A_c, B_k \oplus B_c)$. If $T_{\alpha,\beta}$ contains n predicates, we obtain the large amount of n^9 possible combinations (9-tuples) of topological predicates for the nine pairs. However, many combinations turn out to be invalid for definitional or topological reasons.

We employ so-called *cancellation rules* (Figure 2) to isolate and exclude all invalid predicate combinations. These are based on two principles. First, we consider all predicate combinations that are invalid due to a violation of the definition of vague spatial data types. An example are all combinations for which $inside(A_k, B_k)$ and $contains(A_c, B_k)$ hold, since then $inside(A_k, A_c)$ follows. Second, we take into account those predicate combinations that are invalid due to a contradiction of the predicates involved in the nine relationships. An example are all combinations for which $overlap(A_k, B_k)$ and $disjoint(A_k \oplus A_c, B_k)$ hold. From $disjoint(A_k \oplus A_c, B_k)$ we can conclude $disjoint(A_k, B_k)$ which is a direct contradiction to $overlap(A_k, B_k)$ due to the mutual exclusion of all topological predicates in $T_{\alpha,\beta}$. Since the cancellation rules identify all invalid 9-tuples, all remaining 9-tuples are topologically possible and satisfy the definition of vague spatial data types. Each valid 9-tuple represents a nameless, numbered vague topological predicate.

The large number of vague topological predicates and their fine-grained semantics can overwhelm the user. We therefore propagate an additional evaluation step which rests on the observation that the same predicate *name* can be associated with several 9-tuples, i.e., with different but topologically related spatial configurations. This leads to a grouping of vague topological predicates into so-called *clustered vague topological predicates*. We define *clustering rules* (Figure 2) as type-combination specific evaluation criteria to express whether each valid 9-tuple makes a clustered predicate *true*, *maybe*, or *false*. Clustered predicates are predefined, and different meaningful sets of clustered predicates are conceivable. The clustering rules have the task to define the semantics of each clustered vague topological predicate. The rules for each individual clustered predicate but also for all clustered predicates together must be sound, i.e., mutually exclusive, complete, and cover all valid 9-tuples.

It seems to be natural to take the topological predicates defined on complex crisp spatial data types as a foundation and to define their vague counterparts on the basis of all valid 9-tuples. For example, in Section 5 we deploy the topological predicates $T_{point,point} = \{disjoint, contains, inside, overlap, equal\}$ between two complex crisp point objects. According to our strategy, we obtain the five clustered vague topological predicates $T_{vpoint,vpoint} = \{Disjoint, Contains, Inside, Overlap, Equal\}$, which are indicated by a capital letter and specify their output values in our three-valued logic depending on the clustering rules.

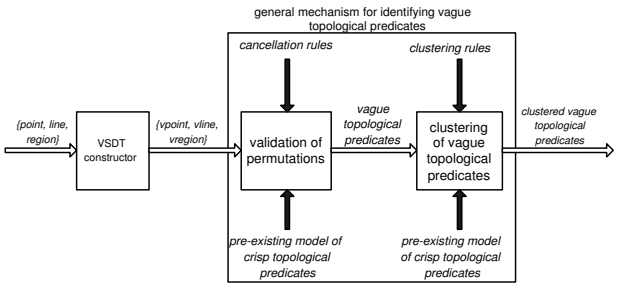

Figure 2: Overview of the general mechanism

5. PREDICATES FOR VAGUE POINTS

In this section, we illustrate the general mechanism of Section 4 by identifying the vague topological predicates and their clustered versions for two *vpoint* objects. Section 5.1 presents a set of *cancellation rules* that extract invalid 9-tuples on the basis of the predefined semantics of the underlying five crisp topological predicates in $T_{point,point}$ [1]. This means we need to explore a total of $5^9 = 1953125$ 9-tuples. Section 5.2 introduces possible *clustering rules* for grouping the valid 9-tuples into the clustered predicates.

5.1 Cancellation Rules

According to Section 4 we distinguish cancellation rules checking the violation of the *vpoint* type definition (Section *5.1.1*) and cancellation rules searching for contradictions of the predicates involved in the nine relationships (Section *5.1.2*). We assume two *vpoint* objects A and B. To make the notation more compact, all rules are parameterized. We introduce the parameters $x, y, i, j \in \{k, c\}$ with $x \neq y$ and $i \neq j$. We also specify parameters $v \in \{A_k, A_c, A_k \oplus A_c\}$ and $w \in \{B_k, B_c, B_k \oplus B_c\}$. Due to a lack of space, we omit all proofs for the rules.

5.1.1 Cancellation by Definition

The following rules are defined on the premise that the kernel part and the conjecture part of a *vpoint* object may only satisfy the topological relationships *disjoint* or *meet* (Section 3). The idea of the first two (symmetric) rules is to invalidate combinations where containment of the interior in one predicate and intersection of the interiors in the other predicate would force the interiors of the individual components of the same object to intersect.

Rule 1 $\forall p \in \{equal, contains\}\, \forall q \in \{equal, overlap, inside, contains\} : \neg(p(A_x, w) \ \wedge \ q(A_y, w))$

Rule 2 $\forall p \in \{equal, inside\}\, \forall q \in \{equal, overlap, inside, contains\} : \neg(p(v, B_i) \ \wedge \ q(v, B_j))$

The next two (symmetric) cancellation-by-definition rules invalidate 9-tuples on the basis that, if a component A_x is inside the union of the components of B but not exclusively inside some component B_i, the other component A_y must not contain the other component B_j. If B_j's interior was inside A_y's interior, A_x's interior would also intersect A_y's interior, since A_x's interior intersects B_j's interior.

Rule 3 $\forall p \in \{overlap, contains\}\, \forall q \in \{equal, contains\}\, \forall r \in \{inside, equal\} : \neg(p(A_x, B_i) \ \wedge \ q(A_y, B_j) \ \wedge \ r(A_x, B_k \oplus B_c))$

Rule 4 $\forall p \in \{overlap, inside\}\, \forall q \in \{equal, inside\}\, \forall r \in \{contains, equal\} : \neg(p(A_x, B_i) \ \wedge \ q(A_y, B_j) \ \wedge \ r(A_k \oplus A_c, B_i))$

In case of two vague point objects, the cancellation-by-definition rules are able to invalidate 1919875 combinations. They leave 33250 valid combinations up to this point.

5.1.2 Cancellation by Contradiction

Cancellation-by-contradiction rules invalidate 9-tuples on the basis of contradictory relationships contained in them. The following two (symmetric) rules are based on the set-theoretic observation that, if two sets X and Y are disjoint, then each subset of X (Y) is disjoint from Y (X) too.

Rule 5 $\forall p \in \{equal, contains, overlap, inside\}$: $\neg(p(A_x, w) \ \wedge \ disjoint(A_k \oplus A_c, w))$

Rule 6 $\forall p \in \{equal, contains, overlap, inside\}$: $\neg(p(v, B_i) \ \wedge \ disjoint(v, B_k \oplus B_c))$

The next two (symmetric) cancellation-by-contradiction rules are also based on a set-theoretic fact: If a set X contains a set Y, then each superset of X also contains Y.

Rule 7 $\forall p \in \{equal, contains\}\ \forall q \in \{disjoint, overlap, inside\} : \neg(p(A_x, w) \ \wedge \ q(A_k \oplus A_c, w))$

Rule 8 $\forall p \in \{equal, inside\}\ \forall q \in \{disjoint, overlap, contains\} : \neg(p(v, B_i) \ \wedge \ q(v, B_k \oplus B_c))$

The next two (symmetric) rules follow the same idea of containment as the previous two rules but reverse the roles by making sure that, if both components of a *vpoint* object are contained inside a component of another *vpoint* object, then their union is also contained.

Rule 9 $\forall r \in \{disjoint, overlap, contains\} : \neg(inside(A_x, w) \ \wedge \ inside(A_y, w) \ \wedge \ r(A_k \oplus A_c, w))$

Rule 10 $\forall r \in \{disjoint, overlap, inside\} : \neg(contains(v, B_i) \ \wedge \ contains(v, B_i) \ \wedge \ r(v, B_k \oplus B_c))$

The following two (symmetric) rules ensure that, if each component of A is disjoint from each component or the union of components of B, the union of components of A is also disjoint from (components of) B. These rules can be considered the opposite of the situations described in the rules 5 and 6.

Rule 11 $\forall r \in \{contains, equal, overlap, inside\}$: $\neg(disjoint(A_x, w) \ \wedge \ disjoint(A_y, w) \ \wedge \ r(A_k \oplus A_c, w))$

Rule 12 $\forall r \in \{contains, equal, overlap, inside\}$: $\neg(disjoint(v, B_i) \ \wedge \ disjoint(v, B_j) \ \wedge \ r(v, B_k \oplus B_c))$

The next two (symmetric) cancellation rules rest on the set-theoretic statement that the non-disjointedness of two sets is maintained if one of the sets is replaced by a superset.

Rule 13 $\forall p \in \{overlap, contains\}\ \forall q \in \{inside, equal\} : \neg(p(A_x, w) \ \wedge \ q(A_k \oplus A_c, w))$

Rule 14 $\forall p \in \{overlap, inside\}\ \forall q \in \{contains, equal\} : \neg(p(v, B_i) \ \wedge \ q(v, B_k \oplus B_c))$

The next two (symmetric) rules assure that, if a component of an object does not contain a component of a second object but the union of the components of the first object contains the component, then the other component of the first object must not be disjoint from the second object.

Rule 15 $\forall p \in \{overlap, disjoint, inside\}\ \forall r \in \{equal, contains\} : \neg(p(A_x, w) \ \wedge \ disjoint(A_y, w) \ \wedge \ r(A_k \oplus A_c, w))$

Rule 16 $\forall p \in \{overlap, disjoint, contains\}\ \forall r \in \{equal, inside\} : \neg(p(v, B_i) \ \wedge \ disjoint(v, B_j) \ \wedge \ r(v, B_k \oplus B_c))$

The last two (symmetric) cancellation-by-contradiction rules deal with the special case that, if an object A is equal to (some component of) another object B but one component of A is disjoint from (some component of) B, the other component of A cannot be disjoint too.

Rule 17 $\forall q \in \{inside, disjoint\} : \neg(equal(A_k \oplus A_c, w) \ \wedge \ q(A_y, w) \ \wedge \ disjoint(A_x, w))$

Rule 18 $\forall q \in \{contains, disjoint\} : \neg(equal(v, B_k \oplus B_c) \ \wedge \ q(v, B_j) \ \wedge \ disjoint(v, B_i))$

The application of all cancellation-by-contradiction rules, which was supported by a utility program not shown here, eliminates further 33053 rules and leaves 197 valid 9-tuples that correspond to vague topological predicates for two *vpoint* objects. All invalid predicate combinations are captured since it is possible for all remaining 197 predicate combinations to draw a spatial configuration representing the nine topological relationships of such a 9-tuple. Due to space restrictions, we are unable to show the valid 9-tuples and their proofs by drawing.

5.2 Clustering Rules

To handle and distinguish such a large amount of very specialized vague topological predicates is difficult for users and not necessarily desired by them. For easier use we therefore group the predicates (Figure 2) into the five *clustered predicates Disjoint*, *Equal*, *Overlap*, *Contains*, and *Inside*, which are supposed to be the vague counterparts of the five topological predicates *disjoint*, *equal*, *overlap*, *contains*, and *inside* between two crisp point objects [1]. *Clustering rules* define for which 9-tuples a clustered predicate yields *true*, *maybe*, and *false*. In the following, we specify these clustering rules for the five clustered predicates. The strategy is always the same. For $p \in T_{vpoint,vpoint}$ we only define the two cases for which $p(A, B) = true$ and $p(A, B) = false$. Then $p(A, B) = maybe \Leftrightarrow \neg(p(A, B) = true \ \vee \ p(A, B) = false)$.

$Equal(A, B)$. Two *vpoint* objects A and B definitely satisfy this predicate if their kernel parts are the same and their conjecture parts are empty (crisp case). The existence of conjecture parts at any rate introduces uncertainty so that equality cannot be assured. If conjecture parts exist, A and B may be equal if the kernel parts are equal, or if one object is contained within the other object or overlaps it and the difference between the objects is all made out of conjecture parts. Finally, A and B are not equal if they are either disjoint or if there is not such containment as just described.

$$
\begin{aligned}
&Equal(A, B) = true \Leftrightarrow \\
&\quad equal(A_k, B_k) \ \wedge \ equal(A_k, B_k \oplus B_c) \ \wedge \\
&\quad equal(A_k \oplus A_c, B_k) \ \wedge \ equal(A_k \oplus A_c, B_k \oplus B_c) \\
&Equal(A, B) = false \Leftrightarrow \\
&\quad overlap(A_k, B_k \oplus B_c) \ \vee \ overlap(A_k \oplus A_c, B_k) \\
&\quad \vee \ inside(A_k \oplus A_c, B_k) \ \vee \ contains(A_k, B_k \oplus B_c) \\
&\quad \vee \ disjoint(A_k \oplus A_c, B_k \oplus B_c) \\
&\quad \vee \ (equal(A_k \oplus A_c, B_c) \ \wedge \ inside(A_k \oplus A_c, B_k \oplus B_c)) \\
&\quad \vee \ (equal(A_c, B_k \oplus B_c) \ \wedge \ contains(A_k \oplus A_c, B_k \oplus B_c)) \\
&\quad \vee \ (overlap(A_k \oplus A_c, B_k \oplus B_c) \\
&\quad \wedge \ \neg(inside(A_k, B_k \oplus B_c) \ \wedge \ contains(A_k \oplus A_c, B_k)))
\end{aligned}
$$

$Inside(A, B)$. A *vpoint* object A is considered certainly inside another *vpoint* object B only if all components of A are inside the kernel part of B. It is uncertain whether A is inside B if A's kernel part is inside B but A's conjecture is not. When the interior of the kernel part of A overlaps the exterior of B we can definitely say that A is not inside B.

$$
\begin{aligned}
&Inside(A, B) = true \Leftrightarrow \\
&\quad inside(A_k \oplus A_c, B_k) \ \wedge \ inside(A_k, B_k) \\
&Inside(A, B) = false \Leftrightarrow \\
&\quad overlap(A_k, B_k \oplus B_c) \ \vee \ contains(A_k, B_k \oplus B_c) \\
&\quad \vee \ disjoint(A_k \oplus A_c, B_k \oplus B_c) \ \vee \\
&\quad (equal(A_k \oplus A_c, B_k \oplus B_c) \ \wedge \ equal(A_k, B_k \oplus B_c))
\end{aligned}
$$

$Contains(A, B)$. This clustered predicate is the inverse version of *Inside*.

$$Contains(A, B) \Leftrightarrow Inside(B, A)$$

$Overlap(A, B)$. Two *vpoint* objects truly overlap only if their kernel parts overlap and there is no possibility of containment once the conjecture parts are considered. The predicate is uncertain if a containment might exist or if there might be an overlapping but not between the two kernel parts. We can assure both objects do not overlap if all components between both objects are certainly disjoint from each other or if any of the other predicates certainly hold.

$$\begin{aligned}
&Overlap(A, B) = true \Leftrightarrow \\
&\quad overlap(A_k, B_k) \ \wedge \ \neg(inside(A_k, B_k \oplus B_c) \\
&\quad \vee \ (equal(A_k, B_k \oplus B_c) \ \wedge \ equal(A_k \oplus A_c, B_k \oplus B_c))) \\
&\quad \wedge \neg(contains(A_k \oplus A_c, B_k) \ \vee \ (equal(A_k \oplus A_c, B_k) \\
&\quad \wedge \ equal(A_k \oplus A_c, B_k \oplus B_c))) \\
&Overlap(A, B) = false \Leftrightarrow \\
&\quad disjoint(A_k \oplus A_c, B_k \oplus B_c) \ \vee \ inside(A_k \oplus A_c, B_k) \\
&\quad \vee \ contains(A_k, B_k \oplus B_c) \ \vee \ (equal(A_k \oplus A_c, B_k \oplus B_c) \\
&\quad \wedge \ (equal(A_k, B_k \oplus B_c) \ \vee \ equal(A_k \oplus A_c, B_k)))
\end{aligned}$$

$Disjoint(A, B)$. Two *vpoint* objects are definitely disjoint if all components from both objects are disjoint from each other. They are perhaps disjoint if we additionally allow that some components but not the kernels of both objects overlap. The predicate is certainly false if the kernel parts of both objects are in a relationship unequal to *disjoint*.

$$\begin{aligned}
Disjoint(A, B) = true \ &\Leftrightarrow \ disjoint(A_k \oplus A_c, B_k \oplus B_c) \\
Disjoint(A, B) = false \ &\Leftrightarrow \ \neg disjoint(A_k, B_k)
\end{aligned}$$

Due to space limitations we are unable to show the clustering of the 197 vague topological predicates of $T_{vpoint,vpoint}$ into the five clustered predicates.

6. QUERYING

In this section, we briefly demonstrate the integration of clustered vague topological predicates into an SQL-like query language. The problem is that we have to transform the results of our three-valued logic into results of the boolean logic common to SQL predicates. As a solution, we assign the three boolean predicates *True_P*, *Maybe_P*, and *False_P* to each clustered predicate $P \in T_{vpoint,vpoint}$. For two vague spatial objects A and B, we obtain:

$$\begin{aligned}
True_P(A, B) = true &\Rightarrow P(A, B) = true \\
True_P(A, B) = false &\Rightarrow P(A, B) = maybe \vee P(A, B) = false \\
Maybe_P(A, B) = true &\Rightarrow P(A, B) = maybe \\
Maybe_P(A, B) = false &\Rightarrow P(A, B) = true \quad \vee P(A, B) = false \\
False_P(A, B) = true &\Rightarrow P(A, B) = false \\
False_P(A, B) = false &\Rightarrow P(A, B) = true \quad \vee P(A, B) = maybe
\end{aligned}$$

Note that $\neg True\_P(A, B) \neq False\_P(A, B)$. Two queries shall illustrate the use of these boolean predicates. We assume a homeland security scenario and store in a database table information about terrorists, the terrorist cell they are associated with, and their known and possible locations modeled as *vpoint* objects. Another table shall contain the description of oil fields where a *vpoint* object specifies the locations of all oil pumps in the field. The first query asks for those fields that have pumps where terrorists might have been located so that it is urgent to secure these oil fields.

```
SELECT f.name
FROM   fields f, terrorists t
WHERE  f.pumps False_Disjoint t.locations
```

The use of the predicate *False_Disjoint* means here that oil pumps and terrorist locations are definitely not disjoint. In other words, the interiors of their kernel parts intersect. For the next query, we supplement the oil field table by another attribute which represents possible, unexploited well locations where oil pumps can be installed. The second query asks for all oil fields that still have possible well locations available for installing oil pumps.

```
SELECT f.name
FROM   fields f
WHERE  f.pumps Maybe_Inside f.wells OR
       f.pumps True_Inside f.wells
```

Note that $Maybe\_Inside(A, B) \ \vee \ True\_Inside(A, B) = \neg False\_Inside(A, B)$ (see definition above).

7. CONCLUSIONS AND FUTURE WORK

Based on a concept of vague spatial data types, a three-valued logic, a set of cancellation rules, and a set of clustering rules, we have presented a general mechanism for identifying vague topological predicates and clustered versions of them. This mechanism is appropriate for all type combinations. Due to the large number of vague topological predicates (197 predicates in the *vpoint*/*vpoint* case) overwhelming users, we have recognized the necessity to group these predicates into a few clustered and manageable predicates (five in the *vpoint*/*vpoint* case).

In the future, we will extend VASA by vague topological predicates for all remaining type combinations according to the general identification mechanism described. Of course, another important topic will then be their efficient implementation on top of existing algebras for crisp spatial data types and predicates.

8. REFERENCES

[1] T. Behr and M. Schneider. Topological Relationships of Complex Points and Complex Regions. *Int. Conf. on Conceptual Modeling*, pp. 56–69, 2001.

[2] P. A. Burrough and A. U. Frank, editors. *Geographic Objects with Indeterminate Boundaries*. GISDATA Series, vol. 2. Taylor & Francis, 1996.

[3] E. Clementini and P. Di Felice. *An Algebraic Model for Spatial Objects with Indeterminate Boundaries*, pp. 153–169. In Burrough and Frank [2], 1996.

[4] A. G. Cohn and N. M. Gotts. *The 'Egg-Yolk' Representation of Regions with Indeterminate Boundaries*, pp. 171–187. In Burrough and Frank [2], 1996.

[5] A. Pauly and M. Schneider. Vague Spatial Data Types, Set Operations, and Predicates. *8th East-European Conf. on Advances in Databases and Information Systems*, 2004.

[6] M. Schneider. *Spatial Data Types for Database Systems - Finite Resolution Geometry for Geographic Information Systems*, volume LNCS 1288. Springer-Verlag, Berlin Heidelberg, 1997.

[7] M. Schneider. Uncertainty Management for Spatial Data in Databases: Fuzzy Spatial Data Types. *6th Int. Symp. on Advances in Spatial Databases*, LNCS 1651, pp. 330–351. Springer-Verlag, 1999.