Implementing Topological Predicates for Complex Regions

Markus Schneider

University of Florida, Department of Computer and Information Science and Engineering, Gainesville, FL 32611, USA, mschneid@cise.ufl.edu

Abstract

Topological predicates on spatial objects have been a main area of research in spatial data handling, reasoning, and query languages. The focus of research, has, to a large extent, been on the design of and reasoning with these predicates, whereas implementation issues have been somewhat neglected. The goal of this paper is to show how an abstract design of topological predicates for *complex* regions can be efficiently implemented. The resulting algorithms are based on the *realm* concept, which is well known in the spatial database community.
Keywords: complex region, topological predicate, implementation, realm, ROSE algebra

1 Introduction

In recent years, significant achievements have been made on the design of topological predicates for spatial objects. Almost all results have been based on either the 9-intersection model (Egenhofer *et al.*, 1990), which rests on point sets and point set topology, or the RCC model (Cui *et al.*, 1993), which employs spatial logic. The foundation for this work is the first model, since the goal of the paper is to address the *efficient* implementation of topological predicates for regions in a spatial database or GIS context.

Whereas the predicates in the aforementioned two models operate on simplified abstractions of spatial objects like simple regions, we are interested in the design and implementation of topological predicates for *complex* regions, which may consist of several components (faces) and which may have holes.

Implementation issues for these predicates, regardless of whether they operate on simple or complex geometries, have so far been somewhat neglected. Hence, it is especially the goal of this paper to demonstrate how an abstract design of topological predicates for complex regions can be efficiently implemented.

Section 2 discusses related work. Section 3 presents an abstract, formal model of topological predicates for complex regions. This model is called *abstract*, because it makes clean and simple definitions in terms of infinite point sets and point set topology, without worrying whether finite representations of these sets exist. Section 4 describes a corresponding discrete model. This model is called *discrete*, because it takes into account *finite representations* only available in computers. In this sense it is nearer to implementation. *Realms* (Güting *et al.*, 1993; Schneider, 1997) will be the basis of this model. Section 5 discusses the implementation level. We will show a realm-based data structure for complex regions and algorithms for the predicates operating on this data structure. Finally, Section 6 draws some conclusions.

2 Related Work

In the past, a number of data models and query languages for spatial data have been proposed with the aim of formulating and processing spatial queries in databases. *Spatial data types* (Schneider, 1997) like *point*, *line*, or *region* are the central concept of these approaches and provide fundamental abstractions for modelling the structure of geometric entities, their relationships, properties, and operations. Whereas in older models the geometric structure of spatial data has been restricted (only simple regions, continuous lines, single points), in the meantime a few models (Güting *et al.*, 1995; Clementini *et al.*, 1996) also allow complex spatial objects which may consist of several disjoint components. Additionally, a region object may have holes, and a component of a line object may be a connected collection of curves. The reasons for defining complex geometric structures are closure properties for spatial operations and application-specific requirements. The OpenGIS Consortium (OGC) has incorporated similar generalised geometric structures, called *simple features*, into their OGC Abstract Specification (OGC, 1999) and into the *Geography Markup Language* (*GML*) (OGC, 2001), which is an XML encoding for the transport and storage of geographic information. These geometric structures are called *MultiPoint*, *MultiLineString*, and *MultiPolygon*. Implementation descriptions for these structures have so far not been given.

A well known, abstract model for characterising topological relationships between simple regions is the 9-*intersection model* (Egenhofer *et al.*, 1989). For two simple regions, eight meaningful configurations have been identified which lead to the eight predicates of the set T_{sr} = {*disjoint*, *meet*, *overlap*, *equal*, *inside*, *contains*, *covers*, *coveredBy*}. Each predicate is determined by a unique 9-intersection matrix representing the nine intersections of the boundary, interior, and exterior of the first region with the corresponding parts of the second region. All predicates are mutually exclusive and cover all topological situations. However, a generalisation of spatial data types also necessitates a generalisation of the corresponding topological predicates. It is surprising that such predicates have until recently not been defined. In (Clementini *et al.*, 1995) the so-called TRCR

(Topological Relationships for Composite Regions) model only allows sets of disjoint simple regions without holes. In (Egenhofer *et al.*, 1994) only topological relationships of simple regions with holes are considered; multi-part regions are not permitted. Recently, a general (Behr *et al.*, 2001) and a limited model (Schneider, 2001) have been presented for topological predicates on complex regions. This paper will be based on the limited model (see Section 3).

So far, implementation issues for topological predicates have been almost completely neglected in the literature. In (Clementini *et al.*, 1994) query processing strategies are discussed for topological predicates on simple regions. The central idea is to find a minimal but still unique subset of the nine intersections for each predicate. Only this subset has then to be evaluated, and the hope is that this reduction leads to a better performance. It is assumed that the computational cost of each of the nine possible intersections is equal. Although this approach is interesting from a theoretical point of view, it is doubtful from an implementation perspective, because the computation of all nine intersections can be and has to be performed by the same algorithmic scheme (see Section 5).

The implementation of spatial objects and predicates is impeded by the assumption of a Euclidean space and an infinite-precision arithmetic in spatial data models which conflicts with the reality of finite-precision number systems available in computers (Schneider, 1997). This leads inevitably not only to numerical but especially to topological errors and thus to wrong query results in database systems. A solution for this problem is the *realm* concept (Güting *et al.*, 1993). A realm replaces the Euclidean space with a discrete geometric basis and is intended to represent the entire underlying geometry of an application. It is based on a finite resolution computational geometry and consists of a finite set of points and *non-intersecting* line segments which are defined over a discrete point grid and which form a spatially embedded planar graph. We will see that on top of realms complex regions and topological predicates operating on them can be easily implemented.

3 The Abstract Model

In this section we review the limited abstract model for topological predicates on complex regions, which has been described in (Schneider, 2001). The objective of this model is not to find all possible topological relationships between two complex regions but to generalise the eight topological relationships for simple regions to complex regions in a straightforward way. This may be regarded as an ad hoc approach leading to too coarse predicates. For many spatial applications, however, this predicate collection is practicable enough, and a more fine-grained differentiation is even not desired.

3.1 Complex Regions

First, we informally describe the structure of complex regions, which results in the spatial data type *region*. A formal definition can be found in (Schneider, 2001) and is based on point set theory and point set topology. Regions are embedded into the two-dimensional Euclidean space IR^2 and are thus point sets.

A *simple region* has a connected interior, a connected boundary, and a single connected exterior. Hence, it does not consist of several components, and it does not have holes.

A *simple region with holes F*, which is also called a *face* (Fig. 1c-e), is a simple region with an outer boundary containing several other simple regions, which are called *holes*. Holes represent areas (simple regions) that do not belong to the region object but are enclosed by it. A hole is allowed to touch the outer boundary of *F* or the boundary of another hole in at most single points. To permit holes to have a partially common border with other holes makes no sense because then adjacent holes could be merged to a single hole by eliminating the common border (similarly for adjacency of a hole with the outer boundary). A face is atomic and cannot be decomposed into two or more faces. For example, the configuration shown in Fig. 1a, must be interpreted as two faces with two holes and not as a single face with four holes.

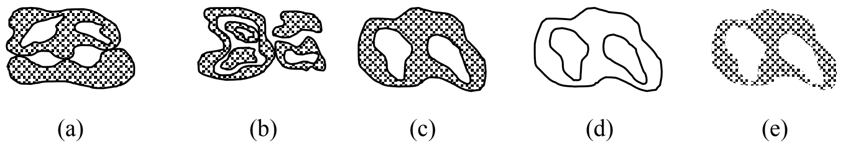

Fig. 1. Unique representation of a face **(a)**, a complex region with five faces **(b)**, a simple region with two holes **(c)**, its boundary **(d)**, and its interior **(e)**

Finally, a *complex region* (Fig. 1b) is a set of faces. A face has to be disjoint to another face, or to meet another face in one or several single boundary points, or to lie within a hole of another face and possibly share one or several single boundary points with the boundary of the hole. Faces having common connected boundary parts with other faces or holes are disallowed. The argumentation is similar to that for the face definition.

3.2 Topological Predicates on Simple Regions with Holes

In the following we use the notation $(P_1|P_2|\ldots|P_n)(F,G)$ as a syntactical simplification for the term $P_1(F,G) \vee P_2(F,G) \vee \ldots \vee P_n(F,G)$ where P_i : *region* $\times$ *region* $\rightarrow$ *bool* is a topological predicate for each $1 \leq i \leq n$.

As a first step to a general definition of topological predicates for complex regions we consider such predicates for simple regions with holes and base their definition on the well known semantics of the topological predicates T_{sr} for simple regions (Section 2), as they have been derived from the 9-intersection model.

Let F and G be two simple regions with holes, F consisting of the simple region with the outer boundary F_0 and the holes $F_1, \dots, F_n$, and G consisting of the simple region with the outer boundary G_0 and the holes $G_1, \dots, G_m$. The topological predicates operating on F and G are defined as follows:

$$\begin{array}{rcl}
disjoint_{srh}(F, G) & := & disjoint(F_0, G_0) \lor \\
& & (\exists\ 1 \le i \le n : inside(G_0, F_i)) \lor \\
& & (\exists\ 1 \le j \le m : inside(F_0, G_j)) \\
meet_{srh}(F, G) & := & meet(F_0, G_0) \lor \\
& & (\exists\ 1 \le i \le n : coveredBy(G_0, F_i)) \lor \\
& & (\exists\ 1 \le j \le m : coveredBy(F_0, G_j)) \\
inside_{srh}(F, G) & := & inside(F_0, G_0) \land \\
& & (\forall\ 1 \le j \le m : disjoint(F_0, G_j) \lor \\
& & \quad (inside(G_j, F_0) \land (\exists\ 1 \le i \le n : inside(G_j, F_i))) \\
contains_{srh}(F, G) & := & inside_{srh}(G, F) \\
equal_{srh}(F, G) & := & equal(F_0, G_0) \land n = m \land \\
& & \exists\ \pi : \{1, \dots, n\} \to \{1, \dots, n\}, \pi \text{ bijective}, \\
& & \forall\ 1 \le i \le n : equal(F_i, G_{\pi(i)}) \\
coveredBy_{srh}(F, G) & := & \neg\ ((inside_{srh}|equal_{srh})(F, G)) \land \\
& & (inside|coveredBy|equal)(F_0, G_0) \land \\
& & (\forall\ 1 \le j \le m : ((disjoint|meet)(F_0, G_j) \lor \\
& & \quad (\exists\ 1 \le i \le n : (inside|coveredBy|equal)(G_j, F_i)))) \\
covers_{srh}(F, G) & := & coveredBy_{srh}(G, F) \\
overlap_{srh}(F, G) & := & \neg\ ((disjoint_{srh}|meet_{srh}|coveredBy_{srh}|covers_{srh}| \\
& & \quad inside_{srh}|contains_{srh}|equal_{srh})(F, G))
\end{array}$$

In (Schneider, 2001) we have shown that the set $T_{srh} = \{disjoint_{srh}, meet_{srh}, coveredBy_{srh}, covers_{srh}, inside_{srh}, contains_{srh}, equal_{srh}, overlap_{srh}\}$ provides a complete coverage of topological relationships for two simple regions with holes, and its elements are mutually exclusive. The set T_{srh} of topological predicates on simple regions with holes is in two ways compatible with the set T_{sr} of topological predicates on simple regions obtained by the 9-intersection model. First, if both F and G do not have holes, then T_{srh} and T_{sr} coincide. Second, each of the eight topological predicates on simple regions with holes has the same boolean results for the nine intersections as the corresponding predicate on simple regions.

3.3 Topological Predicates on Complex Regions

With the aid of the topological predicates on simple regions with holes we are now able to define the corresponding predicates on complex regions. Let F and G be complex regions, F consisting of the simple regions with holes $F_1, \ldots, F_n$, and G consisting of the simple regions with holes $G_1, \ldots, G_m$. We define the following predicates:

$$\begin{array}{rcl}
\mathit{disjoint}_{cr}(F, G) & := & \forall\ 1 \leq i \leq n\ \forall\ 1 \leq j \leq m : \mathit{disjoint}_{srh}(F_i, G_j) \\
\mathit{meet}_{cr}(F, G) & := & \neg \mathit{disjoint}_{cr}(F, G) \wedge \\
 & & (\forall\ 1 \leq i \leq n\ \forall\ 1 \leq j \leq m : \\
 & & \qquad (\mathit{disjoint}_{srh} | \mathit{meet}_{srh})(F_i, G_j)) \\
\mathit{inside}_{cr}(F, G) & := & \forall\ 1 \leq i \leq n\ \exists\ 1 \leq j \leq m : \mathit{inside}_{srh}(F_i, G_j) \\
\mathit{contains}_{cr}(F, G) & := & \mathit{inside}_{cr}(G, F) \\
\mathit{equal}_{cr}(F, G) & := & n = m \wedge \exists\ \pi : \{1, \ldots, n\} \to \{1, \ldots, n\}, \pi \text{ bijective}, \\
 & & \qquad \forall\ 1 \leq i \leq n : \mathit{equal}_{srh}(F_i, G_{\pi(i)}) \\
\mathit{coveredBy}_{cr}(F, G) & := & \neg\ ((\mathit{inside}_{cr} | \mathit{equal}_{cr})(F, G)) \wedge \\
 & & (\forall\ 1 \leq i \leq n\ \exists\ 1 \leq j \leq m : \\
 & & \quad (\mathit{inside}_{srh} | \mathit{coveredBy}_{srh} | \mathit{equal}_{srh})(G_j, F_i)) \\
\mathit{covers}_{cr}(F, G) & := & \mathit{coveredBy}_{cr}(G, H) \\
\mathit{overlap}_{cr}(F, G) & := & \neg\ ((\mathit{disjoint}_{cr} | \mathit{meet}_{cr} | \mathit{coveredBy}_{cr} | \mathit{covers}_{cr} | \\
 & & \quad \mathit{inside}_{cr} | \mathit{contains}_{cr} | \mathit{equal}_{cr})(F, G))
\end{array}$$

With similar arguments as in Section 3.2 we can recognise that two complex regions satisfy exactly one of these topological predicates. In other words, the topological predicates of the set $T_{cr} = \{\mathit{disjoint}_{cr}, \mathit{meet}_{cr}, \mathit{inside}_{cr}, \mathit{contains}_{cr}, \mathit{equal}_{cr}, \mathit{coveredBy}_{cr}, \mathit{covers}_{cr}, \mathit{overlap}_{cr}\}$ are mutually exclusive and complete.

4 The Discrete Model

The discrete model is the interface between the abstract model and the implementation level. It provides a discrete, finite representation based on the realm concept for the complex *region* type of the abstract model with its infinite point set representation. Similarly, the topological predicates of the abstract model are mapped to corresponding predicates of the discrete model and operate now on the finite representations. The discrete model still "abstracts" from implementation aspects (that is, from concrete data structures and algorithms) and gives a definition of the *region* type and the predicates on a discrete geometric basis. Efficiency does not play a role here. The challenge is to ensure numerical robustness (due to rounding errors) and topological consistency of predicates. We show how these requirements can be satisfied. Needed realm concepts are briefly reviewed in Section 4.1. Topological predicates on discrete regions are defined in Section 4.2.

4.1 Needed Realm Concepts

In Section 2 we briefly addressed the problems of implementing spatial objects and topological predicates in a numerically robust and topologically consistent way. Further, we mentioned the *realm* concept (Güting *et al.*, 1993; Schneider, 1997) as a successful approach to solving these problems. In this subsection we will informally give some needed realms definitions. The formal definitions can be found in (Güting *et al.*, 1993).

The underlying space of a realm is a finite discrete grid $N \times N$ with $N = \{0, \ldots, m-1\} \subset$ IN. As primitive objects, points and line segments with co-ordinates in N are defined over this grid. A realm represents a finite, user-definable set of points and *non-intersecting*[1] line segments over this grid and is characterised by the following features: (i) Each point and each end point of a line segment of a realm is a grid point, (ii) each end point of a realm segment is also a point of the realm, (iii) no realm point lies properly within a realm segment, and (iv) no two realm segments intersect except at their end points. There is an obvious interpretation of a realm as a *spatially embedded planar graph* with the realm points as nodes and the realm segments as edges. For the formal realm definition *robust geometric primitives* on realm points and realm segments are needed whose definition can also be found in (Güting *et al.*, 1993). Their essential feature is that they can be implemented in a numerically precise way based on integer arithmetic.

In Section 3.1 we have used a bottom-up strategy at the abstract model to (informally) define "abstract" complex regions, and we have started with *simple regions* as the fundamental and simplest kind of regions. In (Schneider, 2001) all more complex kinds of regions are derived from this notion by *finite* means, because in their definitions we only employ finite quantifications and already known predicate specifications. Hence, at the discrete level, it is sufficient to define the finite analogy of a simple region in the realm context. Discrete versions of simple regions with holes and of complex regions are then straightforward! The analogy of a simple region is called *cycle* alluding to the graph interpretation of a realm and describes the well known linear approximation of a region as a simple polygon. A cycle c is defined by a set of realm segments $S(c) = \{s_0, \ldots, s_{m-1}\}$ such that any two consecutively indexed segments (also s_0 and s_{m-1}) meet, that is, share exactly one common end point, and such that no more than two segments from $S(c)$ meet in any point. Cycle c partitions all grid points into the three finite subsets $P_{in}(c)$ containing the grid points lying inside c, $P_{on}(c)$ containing the grid points lying on c, and $P_{out}(c)$ containing the grid points lying outside c. Let $P(c) :=$

[1]The task of transforming an application's set of *intersecting* line segments into a realm's set of *non-intersecting* line segments is performed by the concept of *redrawing* (Güting *et al.*, 1993).

$P_{in}(c) \cup P_{on}(c)$. Furthermore, let *region* be the type of all discrete, complex regions.

4.2 Topological Predicates on Discrete Regions

Since the topological predicates derived from the 9-intersection model are defined on "abstract" simple regions based on infinite point sets and real numbers, they cannot be directly implemented. However, even if their geometry would be based on floating-point numbers, the simple test whether a point lies on a segment is doomed to fail due to rounding errors. If two computed segments are intended to meet in a common point, a corresponding *meet* predicate will probably yield the wrong result because the equality test of the two common end points is error-prone due to presumably slightly different point representations. In the realm context all these computations are numerically precise and robust and do not cause any problems.

We now give the definitions of the eight topological predicates for simple polygons (cycles), which correspond to the 9-intersection model but are specified on a discrete geometric domain. Let F and G be two simple polygons. Then we define:

$$\begin{aligned}
\mathit{disjoint}(F, G) &:= P(F) \cap P(G) = \varnothing \wedge S(F) \cap S(G) = \varnothing \\
\mathit{meet}(F, G) &:= P_{in}(F) \cap P(G) = \varnothing \wedge P(F) \cap P_{in}(G) = \varnothing \wedge \\
&\quad\ (P_{on}(F) \cap P_{on}(G) \neq \varnothing \vee S(F) \cap S(G) \neq \varnothing) \\
\mathit{equal}(F, G) &:= P(F) = P(G) \wedge S(F) = S(G) \\
\mathit{inside}(F, G) &:= P(F) \subseteq P(G) \wedge P_{on}(F) \cap P_{on}(G) = \varnothing \wedge \\
&\quad\ S(F) \cap S(G) = \varnothing \\
\mathit{contains}(F, G) &:= \mathit{inside}(G, F) \\
\mathit{coveredBy}(F, G) &:= P(F) \subseteq P(G) \wedge \\
&\quad\ (P_{on}(F) \cap P_{on}(G) \neq \varnothing \vee S(F) \cap S(G) \neq \varnothing) \\
\mathit{covers}(F, G) &:= \mathit{coveredBy}(G, H) \\
\mathit{overlap}(F, G) &:= \neg\, ((\mathit{disjoint}|\mathit{meet}|\mathit{coveredBy}|\mathit{covers}| \\
&\quad\quad \mathit{inside}|\mathit{contains}|\mathit{equal})(F, G))
\end{aligned}$$

In Section 3 we have used a bottom-up strategy to define topological predicates on abstract complex regions. We have based their definitions on the predicates on simple regions derived from the 9-intersection model and, as an intermediate step, on the predicates on simple regions with holes. Since the definitions of the predicates on simple regions with holes and on complex regions in addition only employ logical connectives and finite quantifications, we can simply adopt these definitions for topological predicates on discrete complex regions! Consequently, there is not more to do at this point.

5 Data Structure and Algorithms

It is obvious that, in this context, computing with finite points sets is not very efficient in practice. Hence, an efficient data structure for the complex data type *region* (Section 5.1) as well as efficient algorithms for the topological predicates operating on this data structure are needed (Section 5.2). For the data structure a special version of the classical linear boundary representation is used.

5.1 A Data Structure for Complex Regions

Rather than describing the data structure directly in terms of arrays, records, etc., we introduce a higher level description which offers suitable access and construction operations to be used in the algorithms. In a further step, one can then easily design and implement the data structure itself.

Let $N = \{0, \ldots, m-1\} \subset \mathrm{IN}$, and let $P_N = N \times N$ denote the set of all *N-points* (grid points). We define an (x, y)-lexicographical order on P_N, which is defined for $p = (x_1, y_1)$ and $q = (x_2, y_2)$ as $p < q \Leftrightarrow x_1 < x_2 \vee (x_1 = x_2 \wedge y_1 < y_2)$. Let $S_N = P_N \times P_N$ denote the set of *N-segments*. We normalise S_N by the requirement that $\forall s \in S_N : s = (p, q) \Rightarrow p < q$. This enables us to speak of a *left* and a *right end point* of a segment.

The crucial idea for the representation of *region* objects is to regard them as *ordered sequences of halfsegments* (Güting *et al.*, 1995). Let $H_N = \{(s, d) \mid s \in S_N, d \in \{left, right\}\}$ be the set of *halfsegments* where flag d emphasises one of the N-segments' end points, which is called the *dominating point* of h. If $d = left$, the left (smaller) end point of s is the dominating point of h, and h is called *left halfsegment*. Otherwise, the right end point of s is the dominating point of h, and h is called *right halfsegment*. Hence, each N-segment s is mapped to two halfsegments $(s, left)$ and $(s, right)$. Let dp be the function which yields the dominating point of a halfsegment.

For two distinct halfsegments h_1 and h_2 with a common end point p, let α be the enclosed angle such that $0° < \alpha \leq 180°$ (an overlapping of h_1 and h_2 is excluded by the realm properties). Let a predicate *rot* be defined as follows: $rot(h_1, h_2)$ is *true* if and only if h_1 can be rotated around p through α to overlap h_2 in counterclockwise direction. We can now define a complete order on halfsegments which is basically the (x, y)-lexicographical order by dominating points. For two halfsegments $h_1 = (s_1, d_1)$ and $h_2 = (s_2, d_2)$ we obtain:

$$\begin{aligned} h_1 < h_2 \quad \Leftrightarrow \quad & dp(h_1) < dp(h_2) \vee (dp(h_1) = dp(h_2) \wedge \\ & ((d_1 = right \wedge d_2 = left) \vee (d_1 = d_2 \wedge rot(h_1, h_2))) \end{aligned}$$

A value of type *region* (an example is given in Fig. 2) is now defined as an ordered sequence $\langle(h_1, a_1), \ldots, (h_n, a_n)\rangle$ of n halfsegments[2] where each halfsegment h_i has an attached set a_i of *attributes*. Attribute sets are used in algorithms to attach auxiliary information to segments. For example, those halfsegments of a *region* object F carry an associated attribute *InsideAbove* where the area of F lies above or left of its segments.

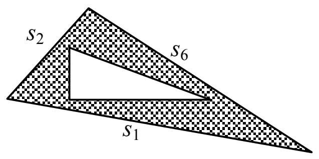

Fig. 2. Example of the halfsegment sequence of a region. If $h_i^l = (s_i, left)$ and $h_i^r = (s_i, right)$ denote the left and right halfsegments belonging to the segments s_i for $1 \leq i \leq 6$, the halfsegment sequence is $\langle(h_1^l, \{I\}), (h_2^l, \varnothing), (h_3^l, \varnothing), (h_4^l, \{I\}), (h_4^r, \{I\}), (h_5^l, \{I\}), (h_2^r, \varnothing), (h_6^l, \varnothing), (h_5^r, \{I\}), (h_3^r, \varnothing), (h_6^r, \varnothing), (h_1^r, \{I\})\rangle$. I stands for the attribute *InsideAbove*.

Simple implementations for type *region* would represent a sequence of n halfsegments in a linked list or sequentially in an array. Unfortunately, in the latter case, inserting a halfsegment at an arbitrary position then needs $O(n)$ time. Alternatively, an AVL-tree embedded into an array can be used whose elements are additionally linked in sequence order. An insertion then requires $O(\log n)$ time.

5.2 Algorithms for Topological Predicates

We now study the algorithms for our set T_{cr} of topological predicates on complex regions. These algorithms are *realm-based* in the sense that the regions as their arguments are defined over the *same* realm, that no two segments intersect within their interiors and that no point lies within a segment. The description of the algorithms requires three important concepts, namely *realm-based plane sweep*, *parallel object traversal*, and *overlap numbers*, which are explained first in the following.

5.2.1 Realm-Based Plane Sweep

The algorithmic scheme we employ for all predicates is the same and is based on the popular *plane sweep* technique (Preparata *et al.*, 1985) of Computational Geometry. A vertical *sweep line* sweeping the plane from left to right stops at

[2]Note that this data structure only stores halfsegments and attributes. The algorithms using this structure are responsible for constructing only halfsegment sequences that maintain the region properties.

special points called *event points* which are generally stored in a queue called *event point schedule*. The event point schedule must allow one to insert new event points discovered during processing; these are normally the initially unknown intersections of line segments. The state of the intersection of the sweep line with the geometric structure being swept at the current sweep line position is recorded in vertical order in a data structure called *sweep line status*. Whenever the sweep line reaches an event point, the sweep line status is updated. Event points which are passed by the sweep line are removed from the event point schedule. Note that in general an efficient fully dynamic data structure is needed to represent the event point schedule and that in many plane-sweep algorithms an initial sorting step is needed to produce the sequence of event points in (x, y)-lexicographical order.

In the special case of realm-based algorithms where no two segments intersect within their interiors, the event point schedule is static (because new event points cannot exist) and given by the ordered sequence of halfsegments of the operand objects. No further explicit event point structure is needed. Also, no initial sorting is necessary, because the plane sweep order of segments is the base representation of *region* objects anyway.

We assume some operations on the sweep line status which later make the description and understanding of the algorithms easier. Each segment stored in this structure is associated with a set of attributes. A new sweep line status structure is initialised by the operation *new_sweep*. If a left (right) halfsegment of a *region* object is reached during a plane-sweep, the operation *add_left* (*del_right*) stores (removes) its segment component into (from) the segment sequence of the sweep line status sorted by the order relation *above*. A segment s lies *above* a segment t if the intersection of their x-intervals is not empty and if for each x of the intersection interval the y-co-ordinate of s is greater than the one of t (except possibly for a common end point where the y-co-ordinates are equal). The operation *pred_exists* (*common_point_exists*) checks whether for the segment currently considered in the sweep line status a predecessor (a neighboureded segment with a common end point) exists. The operation *set_attr* (*get_pred_attr*) sets (gets) a set of attributes for (from the predecessor of) the segment currently considered in the sweep line status.

For the sweep line status an efficient internal dynamic structure like the AVL-tree can be employed which realises the operations *add_left* and *del_right* each in time $O(\log n)$ and the other operations in constant time.

5.2.2 Parallel Object Traversal

Four further operations make the algorithms more readable. The access to the current element of the halfsegment sequence of a *region* object is given by the operation *get_hs*. The operation *get attr* yields the pertaining attribute set of this halfsegment.

For the evaluation of a predicate we have to perform a parallel traversal through the ordered halfsegment sequences of both operand objects. To simplify the description of this parallel scan, two operations are provided. The operation *rr_select_first*(F, G, *object*, *status*) selects the first halfsegment of each of the *region* objects *F* and *G* and positions a logical pointer on both of them. The parameter *object* with possible values {*none*, *first*, *second*, *both*} indicates which of the two object representations contains the smaller halfsegment. If the value of *object* is *none*, no halfsegment is selected, since *F* and *G* are empty. If the value is *first* (*second*), the smaller halfsegment belongs to *F* (*G*). If it is *both*, the first halfsegments of *F* and *G* are identical. The parameter *status* with possible values {*end_of_none*, *end_of_first*, *end_of_second*, *end_of_both*} describes the state of both halfsegment sequences. If the value of *status* is *end_of_none*, both objects still have halfsegments. If it is *end_of_first* (*end_of_second*), *F* (*G*) is empty. If it is *end_of_both*, both object representations are empty. The operation *rr_select_next*(*F*, *G*, *object*, *status*) searches for the next smaller halfsegment of *F* and *G*; parameters have the same meaning as for *rr_select_first*. Both operations together allow one to scan in linear time two object representations like one ordered sequence.

5.2.3 Overlap Numbers

Overlapping of region parts plays an important role for computing their topological relationships. For this purpose we introduce the concept of *overlap numbers*. A point of the realm grid obtains the overlap number 2 if it is covered by (or part of) two *region* objects. This means that for two intersecting simple polygons the area outside of both polygons gets overlap number 0, the intersecting area gets overlap number 2, and the other areas get overlap number 1. Since a segment of a region separates space into two parts, an inner and an exterior one, each segment is associated with a pair (m/n) of overlap numbers, a lower (or right) one m and an upper (or left) one n. The lower (upper) overlap number indicates the number of overlapping *region* objects below (above) the segment. In this way, we obtain a *segment classification* of two *region* objects and speak of (m/n)-segments. Obviously, m, $n \leq 2$ holds. Of the nine possible combinations only seven describe valid segment classes. This is because a (0/0)-segment contradicts the definition of a *region* object, since then at least one of both regions would have two holes or an outer cycle and a hole with a common border. Similarly, (2/2)-segments cannot exist, since then at least one of the two regions would have a segment which is common to two outer cycles of the object. Hence, possible (m/n)-segments are (0/1)-, (0/2)-, (1/0)-, (1/1)-, (1/2)-, (2/0)-, and (2/1)-segments. Examples of (m/n)-segments are given in Fig. 3.

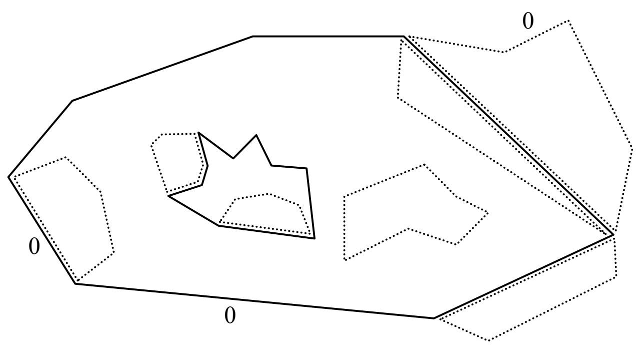

Fig. 3. Example of the segment classification of two *region* objects

5.2.4 Algorithms

The algorithmic scheme, which is common to all topological predicates, is demonstrated by the algorithm for $coveredBy_{cr}(F,G)$. For this predicate all segments of F must lie within the area of G but no segment (and hence no hole) of G may lie within F. If we consider the objects F and G as halfsegment sequences together with the segment classes, the predicate $coveredBy_{cr}$ is *true* if (i) all halfsegments that are *only* elements of F have segment class (1/2) or (2/1), since only these segments lie within G, (ii) all halfsegments that are *only* elements of G have segment class (0/1) or (1/0), since these definitely do not lie within F, and (iii) all *common* halfsegments have segment class (0/2) or (2/0), since the areas of both objects lie on the same side of the halfsegment. We have to require that F and G share at least a common segment or a common point. In the case of a (1/1)-segment the areas would lie side by side so that F could not be covered by G. In the algorithm, whenever a segment is inserted into the sweep line status, first the pair (m_p/n_p) of overlap numbers of the predecessor is determined (it is set to (∗/0) if no predecessor exists). Then the overlap numbers (m_s/n_s) for this segment are computed. Obviously $m_s = n_p$ must hold; n_s is also initialised to n_p and then corrected. We can now formulate the algorithm $coveredBy_{cr}$:

```
algorithm coveredBy_cr
input: Two region objects F and G
output:  true, if F is covered by G
         false, otherwise

begin
  S := new_sweep();
  inside := true;
  common_segment := false;
```

```
    common_point := false;
    rr_select_first(F, G, object, status);
    while (status ≠ end_of_first) and inside do       /* Let h = (s, d). */
      if (object = first) or (object = both) then h := get_hs(F);
      else h := get_hs(G);
      endif;
      if d = right then S := del_right(S, s);
      else
        S := add_left(S, s);
        if not pred_exists(S) then $(m_p/n_p)$ := (*/0)
        else {$(m_p/n_p)$} := get_pred_attr(S);
        endif;
        $m_s := n_p$; $n_s := n_p$;
        if ((object = first) or (object = both)) and (InsideAbove ∈ get_attr(F))
          then $n_s := n_s+1$
          else $n_s := n_s-1$
        endif;
        if ((object = second) or (object = both)) and
              (InsideAbove ∈ get_attr(G))
        then $n_s := n_s+1$
        else $n_s := n_s-1$
        endif;
        S := set_attr(S, {$(m_s/n_s)$});
        if object = first then
          inside := ($(m_s/n_s)$ ∈ {(1/2), (2/1)});
          if common_point_exists(S) then common_point := true endif
        else if object = second then inside := ($(m_s/n_s)$ ∈ {(0/1), (1/0)})
        else inside := ($(m_s/n_s)$ ∈ {(0/2), (2/0)}); common_segment := true
        endif;
      endif;
      rr_select_next(F, G, object, status);
    endwhile;
    return inside and (common_segment or common_point);
  end $coveredBy_{cr}$.
```

If *F* has *l* and *G* *m* halfsegments, the while-loop is executed at most $n = l+m$ times, because each time a new halfsegment is visited. The most expensive operations within the loop are the insertion and the removal of a segment into and from the sweep line status. Since at most *n* elements can be contained in the sweep line status, the worst time complexity of the algorithm is $O(n \log n)$.

The other predicates mostly require slight modifications of the algorithm above. The predicate $covers_{cr}$ is symmetric to $coveredBy_{cr}$. The predicate *insidecr* forbids common (0/2)- and (2/0)- segments and also common points. The same holds for the symmetric predicate $contains_{cr}$. The predicate $disjoint_{cr}$ yields *true* if both objects do not share common areas, common segments, and common points.

Hence, it only allows (0/1)- and (1/0)-segments and forbids common points. The predicate *meet*$_{cr}$ is equal to *disjoint*$_{cr}$ but additionally requires the existence of at least one (1/1)-segment or a common point. The predicate *overlap*$_{cr}$ is *true* if two regions have a common area which means that there exist segments of the segment classes (1/2) **)** or (2/1).

6 Conclusions

Based on a formal and coherent definition of complex regions and their topological predicates both at the abstract and the discrete level, we have demonstrated how regions and predicates on them can be implemented in a numerically robust and topologically consistent manner. Spatial query languages can now also be employed to pose queries using topological relationships on more complex regions.

References

Behr T, Schneider M (2001) Topological Relationships of Complex Points and Complex Regions. In: Int. Conf. on Conceptual Modeling. pp 56-69

Clementini E, Di Felice P (1996) A Model for Representing Topological Relationships between Complex Geometric Features in Spatial Databases. Information Systems 90:121–136

Clementini E, Di Felice P., Califano G (1995) Composite Regions in Topological Queries. Information Systems 20(7):579–594

Clementini E, Sharma J, Egenhofer MJ (1994) Modeling Topological Spatial Relations: Strategies for Query Processing. Computers and Graphics 18(6):815–822

Cui Z, Cohn A G, Randell DA (1993) Qualitative and Topological Relationships. In: 3rd Int. Symp. on Advances in Spatial Databases, LNCS 692, pp 296–315

Egenhofer M J, Frank A, Jackson J P (1989) A Topological Data Model for Spatial Databases. In: 1st Int. Symp. on the Design and Implementation of Large Spatial Databases. LNCS 409, pp 271–286

Egenhofer M J, Herring J (1990) A Mathematical Framework for the Definition of Topological Relationships. In: 4th Int. Symp. on Spatial Data Handling. pp 803–813

Egenhofer MJ, Clementini E, Di Felice P (1994) Topological Relations between Regions with Holes. Int. Journal of Geographical Information Systems 8(2):128–142

Güting R H, Schneider M (1993) Realms: A Foundation for Spatial Data Types in Database Systems. In: 3rd Int. Symp. on Advances in Spatial Databases, LNCS 692, pp 14–35

Güting R H, Schneider M (1995) Realm-Based Spatial Data Types: The ROSE Algebra. VLDB Journal 4:100–143

OpenGIS Consortium (OGC) (1999) OGC Abstract Specification [online]. Available from: http://www.opengis.org/techno/specs.htm.

OpenGIS Consortium (OGC) (2001) OGC Geography Markup Language (GML) 2.0 [online]. Available from: http://www.opengis.net/gml/01-029/GML2.html.

Preparata F P, Shamos M I (1985) Computational Geometry. Springer Verlag

Schneider M (1997) Spatial Data Types for Database Systems–Finite Resolution Geometry for Geographic Information Systems. In: LNCS 1288, Springer-Verlag

Schneider M (2001) A Design of Topological Predicates for Complex Crisp and Fuzzy Regions. In: Int. Conf. on Conceptual Modeling. pp 103-116