Reasoning about Visibility

Roger Villemaire[1a], Sylvain Hallé[b]

[a]Department of Computer Science, Université du Québec à Montréal, C.P. 8888, Succ. Centre-ville, Montréal, Québec, H3C 3P8, Canada
[b]Department of Mathematics and Computer Science, Université du Québec à Chicoutimi, 555 boul. de l'Université, Chicoutimi, Québec, G7H 2B1, Canada

Abstract

We consider properties of sequences of spatial regions, as seen from a viewpoint. In particular, we concentrate on two types of regions: 1) general domains in which a region is any subset of the space, and 2) axis-parallel domains, where the regions are boxes in an N-dimensional space. We introduce binary relations allowing to express properties of these sequences and present two approaches to process them. First, we show that constraints on these relations can be solved in polynomial time for general domain and that the same problem is NP-complete in the axis-parallel case. Second, we introduce a modal logic on these relations, called *Visibility Logic*, and show that model-checking on a finite sequence of regions can be done in polynomial time (both in the general and axis-parallel cases). Finally, we present applications to image processing and firewall filtering.

Keywords: visibility, space, modal logic

1. Introduction

Spatial information and reasoning plays an important role in many application areas such as robotics and geographic information systems. In fact, there are many different facets to spatial information processing. In robotics, one is mainly interested in moving in a two or three-dimensional space, avoiding obstacles toward a destination. In geographic information systems, the main emphasis is in topological relationships between regions such as "inclusion" or "being side-by-side", as is the case for countries having a common border. In this paper, we concentrate on a somewhat different aspect of spatial information, namely object *visibility* according to a viewpoint. We consider *sequences* of spatial regions, and particularly to what extent regions are masked by those appearing before.

We hence aim at describing properties of spatial regions that lie one in front of another. One can think for instance of a computer screen, where the opening of a new window will lay this rectangular region on top, hence hiding parts or whole regions being pushed to the background. Another example is the two-dimensional representation of a three dimensional scene, which is usually done by projecting onto a two-dimensional plane. Here also, regions lying in the foreground hide some parts or even entire regions appearing behind.

Still another application, which we will further develop in Section 5, is network packet filters, such as those used in routers and firewalls. A filter is a sequence of *rules* consisting of two parts: a *condition*, which is a spatial region in the packet space (usually consisting of source and destination IP addresses and TCP ports) and a *decision* (usually to accept or reject the matching packet). Since the first matching rule is applied, a previous rule can (partially) mask a later one.

We therefore consider regions in some space with the additional information about how regions lay (foreground/background) one relatively to another. For instance, in the computer screen example, we

[1]Corresponding author. E-mail addresses: villemaire.roger@uqam.ca (R. Villemaire), shalle@acm.org (S. Hallé).

consider regions on the two-dimensional screen, some regions being (partially) hidden by some others. In the case of a 2D projection of a 3D scene, the regions we consider are the 2D projections of individual 3D regions. Here again other regions appearing in the foreground can hide some parts of some other region. For network packet filters, the regions are the conditions ordered in the way rules appear in the filter.

More specifically, we consider two different types of spatial regions. The first one is *general sets*, where a region is simply given by the enumeration of its elements. The second type is *axis-parallel regions*, where the space is given by a coordinate system and regions are Cartesian products of intervals.

Reasoning about network filters has been considered in a previous paper [44]. Nevertheless apart from the network filter examples, this paper's content is original. For instance, while [44] introduces a first-order formalization and shows that model-checking is PSPACE-complete, we introduce in this paper a modal logic, called *Visibility Logic*, whose model-checking is polynomial time computable. Furthermore we consider in this paper an aspect not considered in [44], namely constraint satisfaction.

This paper is structured as follows. We recall related work in Section 2, before introducing our formalization in Section 3 by giving basic relations and a modal logic for describing visibility properties. Section 4 considers constraint satisfaction and model-checking. We then give in Section 5 some applications of our results to image processing and network filters. Finally Section 6 concludes the paper.

2. Related work

Spatial reasoning has been the focus of related works in a variety of fields. In this section, we survey the most important results related to our contribution.

2.1. *Spatial Reasoning*

Qualitative spatial reasoning has been largely studied particularly in Artificial Intelligence [14, 18, 43]. An influential notion, that of being connected (i.e. to have a common point) has been studied by [16, 17] in a first-order setting. Conveniently, it allows expressing notions such as "to partially overlap", "to be externally connected" or "to be a tangential proper part". This is hence an appropriate setting to describe, for instance, geographical relations of the kind that hold between regions such as countries, provinces, states etc.

Nevertheless, [16, 17] allow distinguishing between open and closed regions, which is not so natural for applications such as Geographical Information Systems. Removing this distinction leads to the introduction of the Region Connection Calculus (RCC) [40]. Unfortunately, RCC turns out to be undecidable [20, 26]. A propositional fragment, called RCC-8, built up from a subset of eight RCC-definable binary relations on regions, was later shown to be decidable [8] and actually NP-complete [41]. Satisfiability in RCC-8 is in fact the constraint satisfaction problem of solving constraints on its 8 relations. Finally [46] extended RCC-8 by adding Boolean operations on regions —yielding BRCC-8— and showed that it is NP-complete (PSPACE-complete for Euclidean spaces).

Contrary to these formalisms, which consider how regions lie one *beside* another, we consider in this paper how they lie one *in front of* another. In fact we don't consider any topological information, such as objects' boundaries meeting, but concentrate our attention on set-theoretical notions and ordering of the regions (see Section 3.1).

Another approach has been [5] that considers interval as a primitive notion, a point of view generalized to dimension 2 by [7]. While we also consider rectangular regions, we do this for any dimension n. Furthermore, instead of topological notions such as intervals meeting (being one immediately after another), we consider the same visibility relations on rectangular regions as on general regions.

Recently [28] introduced a relational framework for occlusion. Their main motivation is computer vision; accordingly the authors consider object fragmentation, which occurs when an occluding object breaks the occluded object into many distinct parts. Contrary to that work, we don't consider fragmentation, but we do introduce relations not considered by these authors, such as globally-obstructs (see Section 3.1).

As in these approaches, we also consider constraint satisfaction problems. We show for instance in Section 4.2.1 that a set of constraint is solvable in polynomial time in the general domain case, while being

NP-complete for axis-parallel domain (Section 4.2.2). Contrary to these approaches, we furthermore extend our relational representation to a full modal logic (see Section 3.2).

The dynamic topological logic of [34] does extend topological considerations to a temporal logic. In that setting the evolution of regions of a topological space is given by the action of a function. Contrary to this approach we neither consider topological notions such as interiors, nor constraint the evolution of our region to be given by some function. We also restrict ourselves to finite sequences and so we don't consider infinite behaviors such as in this work.

2.2. *Spatio-Temporal Reasoning*

Particularly interesting for us is the combination of temporal and spatial logics [1], which is an example of multi-dimensional modal logic [24]. For instance [47, 48] combined *Propositional Temporal Logic* (PTL) [37, 38] —a temporal logic with Next, Since and Until operators, interpreted on $\langle \mathbf{N}, < \rangle$— with BRCC-8. A BRCC-8 formula then describes a relationship between regions at some moment in time.

In this setting, temporal operators can be applied at two quite different levels. For instance applying PTL temporal operators only to BRCC-8 formulas gives a spatio-temporal logic called ST_0. In that setting one can speak of BRCC-8 relations holding between regions at the same moment of time, the PTL temporal connectors controlling which moments are concerned. But, temporal considerations can also be applied to regions themselves. Considering some region C at some instant of time, one can then also speak of this same region at the next instant, denoted by $\bigcirc(C)$. Two further combinations are hence considered by [47, 48]; namely ST_1, which allows applying $\bigcirc$ on regions, and ST_2 which further allows Until, Next (and possibly their past counterparts) on regions. In this setting BRCC-8 relations compare regions at different instant of time.

If one identifies a sequential spatial order going from foreground to background with a time axis going from past to future, a sequence of regions can be considered as being a single region evolving in time. From such a point of view, the Visibility Logic that we develop in Section 3.2 is a spatio-temporal logic considering a single region evolving in time. Compared to the previous spatio-temporal logics, this could seems quite limited as there is not much to be said at a specific instant since one has only a single region! We nevertheless compensate this limitation by considering quite strong relations on the evolution of this single region in time. For instance we consider global-obstruction (Section 3.1), a relation that depends on all previous regions.

3. Formalization

We present in this section our formalization of visibility. We start by defining a set of binary relations suitable for distinguishing visibility-related concepts such as occlusion, obstruction and covering. We then use these binary relations to define a modal logic that we will call Visibility Logic.

3.1. *Visibility relations*

Fix some set $\mathcal{S}$ called the *space*. We consider *regions* of this space, by which we simply mean subsets of $\mathcal{S}$. If we consider two regions in isolation, the major aspects that we consider are the visibility of the background region and to what extent the foreground region obstructs this background region. If A is in the foreground of B, we can distinguish the following three conditions.

- *A occludes B*, when $A \subseteq B$ (Figure 1 a),
- *A obstructs B*, when $A \cap B \neq \emptyset$ (Figure 1 b),
- *A covers B*, when $A \supseteq B$ (Figure 1 c).

Let us denote these relations by $R_\subseteq(A, B)$, $R_\cap(A, B)$ and $R_\supseteq(A, B)$ respectively. Note that these relations are not mere set-theoretical notions, since the A parameter must be in the foreground of B. Therefore for an A in B's foreground, if $R_\subseteq(A, B)$ does not hold (i.e. $\neg R_\subseteq(A, B)$), then $A \setminus B \neq \emptyset$. Similarly, under the

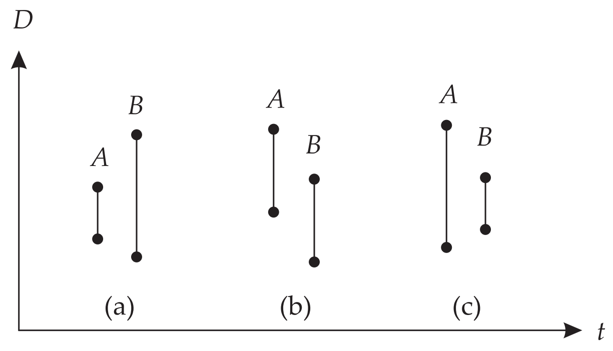

Figure 1: Occludes, obstructs and covers, with foreground on the left.

same condition, if $R_{\supseteq}(A, B)$ does not hold, then $B \backslash A \neq \emptyset$. We can therefore classify the type of relationship that holds between two regions in terms of these three relations. Given two (non-empty) regions A and B, with A in B's foreground, one and only one of the following conditions will hold.

- A *overlaps* B, if $\neg R_{\subseteq}(A, B)$, $R_{\cap}(A, B)$ and $\neg R_{\supseteq}(A, B)$ hold,
- A *strictly covers* B, if $\neg R_{\subseteq}(A, B)$, $R_{\cap}(A, B)$ and $R_{\supseteq}(A, B)$ hold,
- A is *disjoint from* B, if $\neg R_{\subseteq}(A, B)$, $\neg R_{\cap}(A, B)$ and $\neg R_{\supseteq}(A, B)$ hold,
- A *strictly occludes* B, if $R_{\subseteq}(A, B)$, $R_{\cap}(A, B)$ and $\neg R_{\supseteq}(A, B)$ hold,
- A *hides* B, if $R_{\subseteq}(A, B)$, $R_{\cap}(A, B)$ and $R_{\supseteq}(A, B)$ hold (note that in this case $A = B$).

From the three relations, $R_{\subseteq}(A, B)$, $R_{\cap}(A, B)$ and $R_{\supseteq}(A, B)$, one could expect 8 instead of only 5 conditions. But note that the three remaining cases contains $\neg R_{\cap}(A, B)$ with either $R_{\subseteq}(A, B)$ or $R_{\supseteq}(A, B)$ and are therefore void; the intersection of a set with one of its subset always has a non-empty intersection.

The most important aspect of the obstruction concept is that a region A obstructing a region B will make some part of B invisible. Nevertheless in the presence of more than two regions, it is quite possible that removing A would not make this part of B visible either. We therefore have to consider the influence of all regions; we hence introduce the following relation:

- A *globally-obstructs* B (in some spatial sequence), if there is an $a \in A \cap B$ which appears neither in regions appearing before A nor in any region lying between A and B (Figure 2).

We will denote this relation by $R_{\|}(A, B)$. Therefore A globally-obstructs B if A obstructs B and furthermore removing A will make more of B visible. The region A is hence *responsible* for some part of B being invisible.

Note that these relations are based solely on the regions' ordering and pure set-theoretical notions; there is no reference to any topological information.

3.2. *Visibility Logic*

In order to formalize spatial sequence properties, we introduce in this section a modal logic, which we call *Visibility Logic*. Building on Kripke semantics and following the tradition of many-dimensional modal logics [24], we introduce modal operators whose accessibility relations are the binary relations $R_{\subseteq}$, $R_{\cap}$, $R_{\supseteq}$ and $R_{\|}$. But let us first introduce the structures on which our logic will be interpreted.

Since we are considering visibility along a viewpoint, our basic structures are finite sequences of spatial regions, which we will simply name *spatial sequences*. More formally, a spatial sequence $\mathcal{R}$ on a *space* $\mathcal{S}$ is given by a sequence $r_1, \ldots, r_n$ of subsets of $\mathcal{S}$ and sets of propositional variables $\mathcal{P}(r_i)$, $i = 1, \ldots, n$.

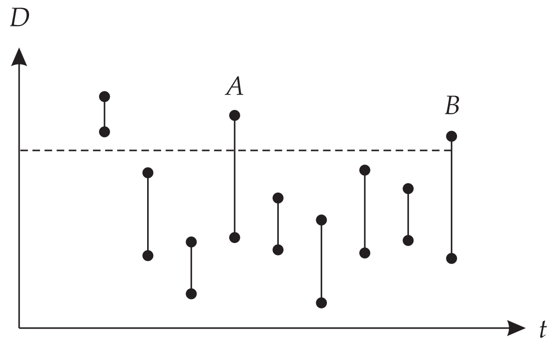

Figure 2: Globally-obstructs, with foreground on the left.

Note that two different regions r_i, r_j ($i \neq j$) of a spatial sequence don't have to be distinct as sets; they could contain exactly the same elements of the space $\mathcal{S}$. A region of a spatial sequence is hence a set in a particular position in this sequence. In order to avoid confusion, we will speak of distinct or different regions for regions appearing in distinct positions, while we will say that two regions have the same *content* if they contain the same elements of $\mathcal{S}$.

We will generally assume that the last region of a spatial sequence is the entire space itself; this will be convenient, as we shall later see.

We define *Visibility Logic* as the multi-modal logic on the binary relations $R_\subseteq$, $R_\cap$, $R_\supseteq$ and $R_\|$. By this we means that there are, for each of these relations R, modal connectives $\bigcirc_R$, $\Diamond_R$ and $\Box_R$. This logic is interpreted on spatial sequences in the following way:

1. $\mathcal{R}, r \models P$, for some propositional variable P, if $P \in \mathcal{P}(r)$,
2. the usual definitions for the Boolean connectives $\neg, \wedge, \vee$,
3. $\mathcal{R}, r \models \bigcirc_R \varphi$, if there is a r' such that $R(r, r')$ and $\mathcal{R}, r' \models \varphi$, for r' the first such region (in sequence order),
4. $\mathcal{R}, r \models \Diamond_R \varphi$, if $\mathcal{R}, r' \models \varphi$, for some region r' satisfying $R(r, r')$,
5. $\mathcal{R}, r \models \Box_R \varphi$, if $\mathcal{R}, r' \models \varphi$, for all regions r' satisfying $R(r, r')$,

Note that in this definition $R(r, r')$ always implies that r appears in the foreground of r'. This means in particular that r and r' are distinct regions of the spatial sequence.

We will furthermore also consider converse connectors $\bigcirc_R^{-1}$, $\Diamond_R^{-1}$ and $\Box_R^{-1}$, in order to speak of previous regions. These connectors have the following semantics.

1. $\mathcal{R}, r \models \bigcirc_R^{-1} \varphi$, if there is a r' such that $R(r', r)$ and $\mathcal{R}, r' \models \varphi$, for r' the last such region (in sequence order),
2. $\mathcal{R}, r \models \Diamond_R^{-1} \varphi$, if $\mathcal{R}, r' \models \varphi$, for some region r' satisfying $R(r', r)$,
3. $\mathcal{R}, r \models \Box_R^{-1} \varphi$, if $\mathcal{R}, r' \models \varphi$, for all regions r' satisfying $R(r', r)$,

To simplify notation, we will write $\bigcirc_\subseteq$, $\Diamond_\subseteq$ and $\Box_\subseteq$ instead of $\bigcirc_{R_\subseteq}$, $\Diamond_{R_\subseteq}$ and $\Box_{R_\subseteq}$. Similarly we will use $\bigcirc_\supseteq$, $\Diamond_\supseteq$, $\Box_\supseteq$, $\bigcirc_\cap$, $\Diamond_\cap$, $\Box_\cap$ and $\bigcirc_\|$, $\Diamond_\|$ $\Box_\|$.

For instance, a region satisfies $\Box_\subseteq \perp$ when it occludes no later region ($\perp$ stands for the *false* constant). A region satisfies $\Diamond_\cap \top$ if it obstructs some later region. Finally a region satisfies $\Diamond_\supseteq^{-1} \top$, if this region is covered by some previous region.

Since we assume that the last region of a spatial sequence is the entire space itself, we have that a region r satisfies $\Diamond_\| \top$ exactly when there is a point of r not contained in any previous region. The region r is then the *first matching region* for this point.

Let us note that there are interactions between Visibility Logic's modalities. By this we mean that there are Visibility Logic formulas, involving the modalities in a non-trivial way that are satisfied by every region of every spatial sequence. While a complete description of these interactions would lead to a complete axiomatization of Visibility Logic, a question we won't further analyze in this paper, the next result gives an example of such formulas.

Proposition 1. *The following Visibility Logic formula is satisfied by every region of every spatial sequence.*

$$(\Diamond_{\subseteq}\varphi) \wedge (\Diamond_{\supseteq}\psi) \rightarrow ((\Diamond_{\subseteq}\Diamond_{\supseteq}\psi) \vee (\Diamond_{\supseteq}\Diamond_{\subseteq}\varphi) \vee ((\Diamond_{\subseteq}(\varphi \wedge \psi)) \wedge (\Diamond_{\supseteq}(\varphi \wedge \psi))))$$

Proof. The result follows from the fact that a spatial sequence is linearly ordered.

Indeed, take $\mathcal{R}, r \models (\Diamond_{\subseteq}\varphi) \wedge (\Diamond_{\supseteq}\psi)$. It follows from the semantics that there are r_1, r_2 appearing in r's background (in $\mathcal{R}$), such that $r \subseteq r_1$, $r \supseteq r_2$ and furthermore $\mathcal{R}, r_1 \models \varphi$ and $\mathcal{R}, r_2 \models \psi$ hold. By transitivity of inclusion it follows that $r_1 \supseteq r_2$. Since $\mathcal{R}$ is linearly ordered, either r_1 appears in r_2's foreground, r_2 appears in r_1's foreground or r_1 and r_2 are the same region.

In the first case $\mathcal{R}, r_1 \models \Diamond_{\supseteq}\psi$, since $\mathcal{R}, r_2 \models \psi$. We therefore have that $\mathcal{R}, r \models \Diamond_{\subseteq}\Diamond_{\supseteq}\psi$. In the second case, $\mathcal{R}, r_2 \models \Diamond_{\subseteq}\varphi$, since $\mathcal{R}, r_1 \models \varphi$ and therefore $\mathcal{R}, r \models \Diamond_{\supseteq}\Diamond_{\subseteq}\varphi$ holds. In the third case r_1 and r_2 are the same region and we therefore have both $\mathcal{R}, r \models \Diamond_{\subseteq}(\varphi \wedge \psi)$ and $\mathcal{R}, r \models \Diamond_{\supseteq}(\varphi \wedge \psi)$, since $\mathcal{R}, r_1 \models \varphi$ and $\mathcal{R}, r_2 \models \psi$ hold.

All together, we have that $\mathcal{R}, r \models (\Diamond_{\subseteq}\Diamond_{\supseteq}\psi) \vee (\Diamond_{\supseteq}\Diamond_{\subseteq}\varphi) \vee ((\Diamond_{\subseteq}(\varphi \wedge \psi)) \wedge (\Diamond_{\supseteq}(\varphi \wedge \psi)))$, showing that $\mathcal{R}, r \models (\Diamond_{\subseteq}\varphi) \wedge (\Diamond_{\supseteq}\psi) \rightarrow ((\Diamond_{\subseteq}\Diamond_{\supseteq}\psi) \vee (\Diamond_{\supseteq}\Diamond_{\subseteq}\varphi) \vee ((\Diamond_{\subseteq}(\varphi \wedge \psi)) \wedge (\Diamond_{\supseteq}(\varphi \wedge \psi))))$, for any $\mathcal{R}$ and r. $\square$

4. Constraint satisfaction and model-checking

We consider in this section two problems. First the satisfiability of a set of constraints on the predicates $R_{\subseteq}$, $R_{\cap}$, $R_{\supseteq}$, $R_{\|}$ and their negations and secondly the model-checking problem, which is to check the validity of a Visibility Logic formula on a particular region of a specific spatial sequence.

We first start with the constraint satisfaction problem (CSP) on the predicates $R_{\subseteq}$, $R_{\cap}$, $R_{\supseteq}$, $R_{\|}$ and their negations. We hence consider a finite sequence of variables $X_1, \ldots, X_n$ and a set of constraints on these variables and we want to assign regions to these variables, in such a way as to fulfill the given constraints.

We consider two types of region representation. First, the *general domain* case in which any subset of the space is a possible region; accordingly, a region is represented by the list of its elements. Second, the *axis-parallel* case where a region is an axis-parallel box. In this case the space is a finite Cartesian product $\prod_{i=1}^{d} \mathcal{D}_i$ of linearly ordered sets $\mathcal{D}_i = \langle D_i, < \rangle$; a region being of the form $r = \prod_{i=1}^{d}[l_i, u_i]$, where $[l_i, u_i]$ is the closed interval $\{x \in D_i; l_i \leq x \leq u_i\}$. In this last case a region is represented by the list of its intervals' bounds $(l_1, u_1), \ldots, (l_d, u_d)$.

For both representations, a spatial sequence is represented by the list of its regions and sets of propositional variables labeling these regions. We will now first show that checking whether a set of constraints is satisfied by a specific assignment can be done in polynomial time.

4.1. Constraint checking

We consider in this section a spatial sequence and an assignment of variables to its regions. We will show that a constraint on these variables can be verified in time polynomial in the size of the spatial sequence. We will hence have that a set of constraints can be verified in time polynomial in the size of the spatial sequence and size of the set of constraints.

Let us first consider the general domain case.

Proposition 2. *Let A and B be regions of a spatial sequence $\mathcal{R}$. In the general domain case the relations $R_{\subseteq}(A, B)$, $R_{\cap}(A, B)$, $R_{\supseteq}(A, B)$ and $R_{\|}(A, B)$ can be checked in time quadratic in the size of $\mathcal{R}$.*

Proof. Indeed in the general domain case the relations $R_{\subseteq}(A, B)$, $R_{\cap}(A, B)$ and $R_{\supseteq}(A, B)$ can be checked in quadratic time by pairwise comparing the elements of A and B. As for relation $R_{\|}(A, B)$, one can simply scan the complete spatial sequence for each element of A, verifying in what other regions it appears. This can be done again in quadratic time. $\square$

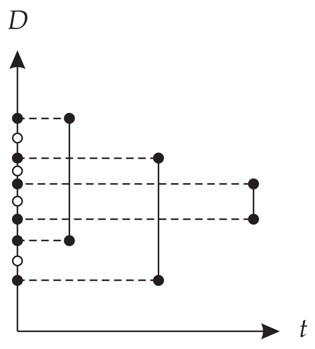

Figure 3: Choosing a subset of the domain based on each region's endpoints.

Similarly, in the axis-parallel domain case, we have the following result.

Proposition 3. *Let A and B be regions of a spatial sequence $\mathcal{R}$. In the axis-parallel domain case the relations $R_{\subseteq}(A,B)$, $R_{\cap}(A,B)$ and $R_{\supseteq}(A,B)$ can be checked in time quadratic in the size of $\mathcal{R}$. As for $R_{\|}(A,B)$, it can be checked in time $O(N^d)$ where N is the size of $\mathcal{R}$ and d the dimension of the axis-parallel space.*

Proof. In the axis-parallel domain case, the relations $R_{\subseteq}(A,B)$, $R_{\cap}(A,B)$ and $R_{\supseteq}(A,B)$ can be checked in quadratic time, by comparing bounds. As for relation $R_{\|}(A,B)$, one can check whether there is an element of $A \cap B$, which is in no region standing either before A or between A and B by the following method.

One has to find a $(x_1,\ldots,x_d) \in \prod_{i=1}^d D_i$, which is in $A \cap B$ while being in no region standing either before A or between A and B. Let us now show that we can substantially limit the number of tuples to consider. Take D'_i to be a subset of D_i containing all the regions' interval endpoints (in coordinate i) occurring in the spatial sequence, plus for any such consecutive endpoints an additional element located between them (if there is one in D_i). Figure 3 illustrates this construction.

We have that given some tuple $(x_1,\ldots,x_d) \in \prod_{i=1}^d D_i$, there is always a $(y_1,\ldots,y_d) \in \prod_{i=1}^d D'_i$ such that all pairs x_i, y_i compare in the same way with all regions' interval endpoints (in coordinate i) occurring in the spatial sequence. We therefore have that $(x_1,\ldots,x_d)$ and $(y_1,\ldots,y_d)$ are in exactly the same sequence's regions. It is therefore sufficient to restrict the search to $\prod_{i=1}^d D'_i$, which is of size $O(N^d)$, showing the claim. □

4.2. *Constraint satisfaction*

Given a set of constraints over a finite sequence of variables $X_1,\ldots,X_n$, we now ask whether it is possible to find a spatial sequence and assign regions to these variables, in such as way as to fulfill the given constraints.

We will show in this section that this problem's complexity depends on the case considered. Namely constraint satisfaction can be done in polynomial time in the general domain case, while it is NP-complete in the axis-parallel domain case.

4.2.1. *General domain case*

We will show in this section that constraint satisfaction can be done in the general domain case in polynomial time. In order to show this, we introduce a *witness* procedure allowing to translate this problem into propositional logic. While propositional logic satisfiability is a well-known NP-complete problem, we will show that our translation produces a specific *conjunctive normal form* (CNF) formula, solvable in polynomial time.

Given a finite sequence of variables $X_1,\ldots,X_n$ and a set of constraints on the predicates $R_{\subseteq}$, $R_{\cap}$, $R_{\supseteq}$, $R_{\|}$ and their negations, we produce a propositional formula that has a solution exactly when it is possible to assign regions to these variables, in such as way as to fulfill the given constraints. Note that $X_1, \ldots, X_n$'s order must be preserved. Therefore for $i < j$, X_i must be assigned to a region appearing before X_j.

We refer to [33] as a general reference on propositional logic. Let us recall that a *clause* is a disjunction of literals, which are themselves either variables or negations of variables. A set of clauses is usually called a SAT-*instance* or simply an instance. For l a literal, we represent by $\bar{l}$ its *complement*, which is $\neg l$ if l is a variable and x if l is the negation $\neg x$ of a variable x.

Our propositional variables will be of the form $d \in X_i$, where X_i is a (region) variable while d is taken from a set of witnesses constructed during the procedure.

We divide the constraints in two sets. First $\neg R_{\subseteq}$, $R_{\cap}$, $\neg R_{\supseteq}$, $R_{\|}$ and secondly $R_{\subseteq}$, $\neg R_{\cap}$, $R_{\supseteq}$, $\neg R_{\|}$.

From the first group, we build a set D, by introducing for each constraint of this group a new element $d \in D$ and adding the following clauses.

- for $\neg R_{\subseteq}(X_i, X_j)$, add $d \in X_i$ and $d \notin X_j$,
- for $R_{\cap}(X_i, X_j)$, add $d \in X_i$ and $d \in X_j$,
- for $\neg R_{\supseteq}(X_i, X_j)$, add $d \in X_j$ and $d \notin X_i$,
- for $R_{\|}(X_i, X_j)$, add
  - $d \notin X_{i'}$, for all $i' < i$,
  - $d \in X_i$,
  - $d \notin X_{j'}$ for all j' such that $i < j' < j$,
  - and finally add $d \in X_j$.

The previous clauses ensure that d is indeed a witness of the fact that the corresponding constraint is satisfied. For instance in the first case, the two clauses ensure that X_i has an element not in X_j. In the second case, the clauses ensure the presence of an element both in X_i and X_j. In the third case the two clauses assure that X_j has an element not in X_i. Finally in the last case, the clauses ensure the presence of an element both in X_i and X_j but in no X_k appearing either before X_i or between X_i and X_j.

We translate the second group of constraints into the following clauses.

- for $R_{\subseteq}(X_i, X_j)$, add $\bigwedge_{d \in D}(d \in X_i \to d \in X_j)$,
- for $\neg R_{\cap}(X_i, X_j)$, add $\bigwedge_{d \in D}(d \in X_i \to d \notin X_j)$
- for $R_{\supseteq}(X_i, X_j)$, add $\bigwedge_{d \in D}(d \in X_j \to d \in X_i)$,
- for $\neg R_{\|}(X_i, X_j)$, add $\bigwedge_{d \in D}((\bigvee_{i' < i} d \in X_{i'}) \vee d \notin X_i \vee (\bigvee_{j' : i < j' < j} d \in X_{j'}) \vee d \notin X_j)$.

This second group contains clauses parameterized by the element of the set D which was generated by the first group of clauses. It must therefore be generated after the first one.

Once again, the clauses ensure that the elements of D fulfill the conditions imposed by the constraint. For instance, the first case ensures that every d that is in X_i is also in X_j. In the second case each d in X_i cannot be also in X_j, which is equivalent to having an empty intersection. The third case is similar to the first. The last case express the fact that there are no d both in X_i and X_j while not being in any region appearing before X_i or between X_i and X_j.

Note finally that these clauses are purely formal Boolean clauses on propositional variables of the form $d \in X_i$. The next result shows the important fact that there is an equivalence between the existence of a spatial sequence and assignment of variables to regions satisfying a set of constraints and satisfiability of the corresponding clauses.

Proposition 4. *The above construction of a set of clauses from a set of constraints has the following properties. If there is a spatial sequence and assignment of variables to regions satisfying the set of constraints then the clauses are satisfiable. Conversely, if the clauses are satisfiable then there is a spatial sequence and assignment of variables to regions satisfying the set of constraints*

Proof. Take a spatial sequence and an assignment satisfying the set of constraints. For each of the constraints of the first group, it is possible to pick an element d satisfying the corresponding clause, where $d \in X_i$ is interpreted as d is in the region assigned to X_i. This will give a truth assignment for the propositional variables $d \in X_i$. Furthermore all clauses of the second group corresponding to a constraint of our set will be satisfied by this assignment. We hence have a satisfying assignment for the complete set of clauses.

Conversely, if one has a satisfying assignment α for the clauses associated to a set of constraints, then assigning the variable X_i to the region $\{d \in D; \alpha(d \in X_i) = \top\}$, i.e. the set of witnesses d such that $d \in X_i$ is assigned to true, will give a spatial sequence fulfilling the set of constraints. $\square$

Now in order to show that the satisfiability of a set of constraints can be checked in polynomial time, it is sufficient to show that any set of clauses of the above type can be checked in polynomial time. But since the satisfiability of Boolean clauses is a well-known NP-complete problem, one must show that the above clauses are of particular type, allowing polynomial satisfiability.

Altogether, the produced clauses are either unary (containing a unique literal) or binary (containing two literals) except for the last clauses, which were produced for $\neg R_{\|}$. Since satisfiability of a CNF containing only unary and binary clauses (2-SAT) is well known to be a polynomially solvable problem, one may wonder whether this complexity can be extended to the complete set of clauses. We will now show that this is indeed the case. In order to show this, we first need to analyze the specific nature of the clauses produced for $\neg R_{\|}$.

We will consider an extension of the SAT problem where the Boolean variables are totally ordered. We therefore consider an ordered sequence $x_1, \dots x_m$ of Boolean variables and relatively to this total order, we define the following notions.

Definition 1. *An* initial *clause, relatively to $x_1, \dots x_m$ is a clause whose literals form an initial segment. This means that if x_i or its negation appears in the clause, then each variable x_j with $j < i$ must also appear in this clause (either positively or negatively).*

Definition 2. *A* spanning *clause is an initial clause with at most two negative literals, such that if it contains exactly two negative literals, its last literal is negative.*

Note that the notions of initial and spanning clause are always relative to a fixed given total order on the variables. In order to apply these notions to our generated clause, first note that there are never two different d's in the same clause. We can therefore split an instance formed of these clauses into one instance for each d, regrouping together all clauses containing a variable of the form $d \in X_i$ for a fixed d. Now the original instance is satisfiable if and only if all these instances are. We can therefore check each of these instances for satisfiability independently.

For the remaining of this section, fix a d and consider the instance formed of the clauses containing $d \in X_i$ for the various X_i. We consider the total order on these variables given by the sequence $X_1, \dots, X_n$, i.e. $(d \in X_i) < (d \in X_j)$ if and only if $i < j$.

Note that the clauses produced for $\neg R_{\|}$ in the above translation are spanning. We hence have that all clauses produced by the above translation are unary, binary or spanning. It is now sufficient to show that satisfiability of such a set of clauses can be checked in polynomial time.

2-SAT is a well-known polynomially solvable problem. In fact [6] has shown how it can be solved in linear-time using a graph theoretical representation. Let us recall some fundamental facts about this solution, which will be useful in solving instances containing spanning clauses.

First as any unary clause l can be transformed into the equivalent binary clause $l \vee l$, we don't have to specifically consider unary clauses. For $\mathcal{I}$ a SAT-instance (a set of clauses) [6] considers the directed graph $G(\mathcal{I})$, whose vertices are the set of literals and whose arcs are given by the binary clauses of $\mathcal{I}$. More precisely, $G(\mathcal{I})$ contains, for every variable v appearing in $\mathcal{I}$, two vertices v and $\bar{v}$. Furthermore if $l \vee l'$ is a binary clause of $\mathcal{I}$, the graph $G(\mathcal{I})$ contains the arcs $\bar{l} \to l'$ and $\bar{l'} \to l$. Note that these two arcs can also be considered as logical implications equivalent to the clause $l \vee l'$.

$G(\mathcal{I})$ has the important property that the permutation mapping literal l to its complement $\bar{l}$ sends an arc to a reverse arc of $G(\mathcal{I})$. For instance, $l \to l'$ will be mapped to $\bar{l'} \to \bar{l}$.

Note also that an assignment α of truth-values to literals (which maps a literal's complement $\bar{l}$ to the complement of $\alpha(l)$), satisfies all binary clauses of $\mathcal{I}$ exactly when $G(\mathcal{I})$ contains no arc $l \to l'$ such that $\alpha(l) = TRUE$ and $\alpha(l') = FALSE$.

In order to check satisfiability of an SAT-instance B containing only binary clauses, [6] considers the decomposition of $G(\mathcal{I})$ in *strongly connected components*.

Let us recall that, in a directed graph, a strongly connected component is a maximal set of vertices C such that for any vertices $v, w \in C$ there is a directed path from v to w. The set of strongly connected components forms a partition of the graph's vertices, i.e. strongly connected components are disjoint and any vertex is in some strongly connected component (where it can be the only element). Finally the set of strongly connected components are the vertices of a directed acyclic graph (DAG), whose arcs are $S \to S'$ for components S and S' such that $s \to s'$ for some (or equivalently all) $s \in S$ and $s' \in S'$.

For S a strongly connected component of $G(\mathcal{I})$ let us denote by $\bar{S}$ the set $\{\bar{s}; s \in S\}$. Note that in the case of $G(\mathcal{I})$, if $S \to S'$ for two strongly connected components S and S', it follows that $\bar{S}' \to \bar{S}$.

The main result about 2-SAT satisfiability is the following theorem of [6].

Theorem 1 ([6, Theorem 1])**.** *A set of binary clause $\mathcal{I}$ is satisfiable if and only if no literal l is in the same connected component as its complement $\bar{l}$ in the graph $G(\mathcal{I})$.*

The linear-time algorithm of [6] first uses Tarjan algorithm [42] to compute strongly connected component and then checks satisfiability using Theorem 1. Furthermore, using the fact that Tarjan's algorithm lists the strongly connected components in inverse topological order, which means that if C appears before C' there are no arc $C \to C'$, [6] shows how to construct a satisfying assignment when the instance is indeed satisfiable.

We will now show that a similar argument can be used to check the satisfiability of SAT-instances containing only binary and spanning clauses. But before let us see to which extend a CNF can be simplified using its unary and binary clauses.

First note that a list of $G(\mathcal{I})$'s strongly connected components in inverse topological order allows to check the following conditions and apply the corresponding simplification.

1. **Contradictory component:** If a strongly connected component of $G(\mathcal{I})$ contains both a literal l and its complement $\bar{l}$, the instance $\mathcal{I}$ is unsatisfiable.
2. **Implication chain removal:** If $G(\mathcal{I})$ contains a directed path from some literal l to its complement $\bar{l}$, then an assignment satisfying $\mathcal{I}$ has to map l to $FALSE$. Accordingly any clause containing $\bar{l}$ can be removed from $\mathcal{I}$ and l can be remove from all clauses.

Note that in the case of a contradictory component, with a table mapping a vertex to its component, one can check whether l and $\bar{l}$ are in the same component in constant time. Checking whether there exists a literal in the same component as its complement can then be done in time linear in the size of the instance.

In the case of an implication chain, a depth-first traversal of the connected components' DAG will allow to find all literals l having a directed path to their complements. Since removing clauses from the instance and literals from clauses can also be done in linear time, the total complexity is linear in the size of the instance.

Let us say that an instance $\mathcal{I}$ *contains no contradictory component* if no strongly connected component of $G(\mathcal{I})$ contains both a literal l and its complement $\bar{l}$. Similarly let us say that $\mathcal{I}$ *contains no implication chain* if $G(\mathcal{I})$ contains no directed path from some literal l to its complement $\bar{l}$. Finally let an l-clause be a clause containing the literal l. The algorithm of Figure 4 and Proposition 5 are implicit in [6].

Proposition 5. *Let $\mathcal{I}$ be an SAT-instance containing no contradictory component nor implication chain. Suppose also that there is a literal l contained in all non-binary clauses of $\mathcal{I}$. Then $\mathcal{I}$ is satisfiable. Furthermore the algorithm of Figure 4 will find a satisfying assignment in time linear in the size of $\mathcal{I}$.*

Proof. First note that the algorithm of Figure 4 runs in time linear in the size of $\mathcal{I}$. This follows from the fact that loops of lines 9 and 13 can be executed in linear-time by depth-first traversals and that furthermore each strongly connected component is considered only once in the loop starting at line 15.

```
1: function $LBinSat(\alpha, \mathcal{I})$
2: Precondition: $\alpha$ is an empty assignment,
3:               $\mathcal{I}$ is a SAT-instance containing no contradictory component,
4:               nor implication chain. Furthermore there is a literal $l$ contained
5:               in all non-binary clauses of $\mathcal{I}$.
6: Postcondition: $\alpha$ is a satisfying assignment for $\mathcal{I}$.
7: begin
8:    $\alpha(l) := TRUE; \alpha(\bar{l}) := FALSE$
9:    for all directed path from $l$ to $l'$
10:   do
11:       $\alpha(l') := TRUE$
12:   for all directed path from $l'$ to $\bar{l}$
13:   do
14:       $\alpha(l') := FALSE$
15:   for all strongly connected components $C$ (in inverse topological order)
16:   do
17:       if $\alpha$ is not defined on $C$
18:       then $\alpha(C) := TRUE$
19:            $\alpha(\bar{C}) = FALSE$
20: end
```

Figure 4: Satisfying assignment for binary and l-clauses

Note also that no literal l will be assigned both at line 11 and 14. Indeed, if $\alpha(l')$ was assigned at both of these lines we would have some path from l to l' and also some path from l' to $\bar{l}$. Merging these path would give an implication chain from l to $\bar{l}$, contradicting the hypothesis.

Note that these assignments make sure that if α is defined on some literal, it is defined for every literal of its strongly connected component. In order to complete the definition of α, the loop at line 15 define α on the remaining components. We use $\alpha(C) := TRUE$ as a short hand for the assignment of all $\alpha(c)$ ($c \in C$) to $TRUE$ (similarly for $\alpha(\bar{C}) = FALSE$). This is the same method as the one used in the proof of [6, Theorem 1]. Note that the remaining strongly connected components are considered in inverse topological order.

It remains to be shown that α satisfies $\mathcal{I}$. First note that since $\alpha(l) = TRUE$, all non-binary clauses of $\mathcal{I}$ are satisfied. For the remaining clauses to be satisfied, it is sufficient to show that there is no arc $l' \to l''$ in $G(\mathcal{I})$ such that $\alpha(l') = TRUE$ and $\alpha(l'') = FALSE$.

This can be showed in the following way. In such a case, $\alpha(l'') = FALSE$ cannot have been set on line 14, since if there is a path from l'' to $\bar{l}$, there is one from l' to $\bar{l}$ and hence $\alpha(l')$ would have been also set to $FALSE$ on the same line.

We therefore have that l'''s connected component L'' was considered in the loop of line 15 and since $\alpha(l'') = FALSE$, this occurred after the assignment $\alpha(\bar{L}'') := TRUE$. It follows that $\bar{L}''$ occurs before L'' in inverse topological order.

Similarly $\alpha(l')$ was defined in the loop of line 15 and its strongly connected component L' occurs before $\bar{L}'$ is inverse topological order. Now since components are visited in inverse topological order and $l' \to l''$, it follows that L'' occurs before L' in this order. We therefore have that $\bar{L}''$ occurs before $\bar{L}'$.

But this is impossible since from $L' \to L''$ it follows that $\bar{L}'' \to \bar{L}'$ contradicting the fact that $\bar{L}''$ occurs before $\bar{L}'$ in inverse topological order. □

Before we can give a satisfiability algorithm for binary and spanning clauses (Figure 5) and its correctness proof (Proposition 7), we need the next simple proposition.

```
1: function $SpanSat(\alpha, \mathcal{I})$
2: Input: $\mathcal{I}$ a set of binary and spanning clauses.
3: Output: Returns $SAT$ and a satisfying assignment $\alpha$ if $\mathcal{I}$ is satisfiable and $UNSAT$ otherwise.
4: begin
5:    $\alpha := \emptyset$
6:    if $\mathcal{I} = \emptyset$, then return $SAT$
7:    else if $\mathcal{I}$ contains an empty clause, then return $UNSAT$
8:          else remove implication chains from $\mathcal{I}$
9:                if $\mathcal{I}$ contains a contradictory component then return $UNSAT$
10:               else $\mathcal{J} := \mathcal{I}$ // make of copy of $\mathcal{I}$ for later use.
11:                     take $x_i$ to be the first variable in $\mathcal{I}$.
12:                     $\alpha(x_i) := TRUE$
13:                     if removing implication chains leads to contradictory component
14:                     then $\mathcal{I} := \mathcal{J}$ // undo $\alpha(x_i) := TRUE$ and implication chain removal.
15:                           $\alpha(x_i) := FALSE$
16:                           return $SpanSat(\mathcal{I})$;
17:                     else $LBinSat(\alpha, \mathcal{I})$;
18:                           return $SAT$;
19: end
```

Figure 5: Satisfiability algorithm for binary and spanning clauses

Proposition 6. *Let $\mathcal{I}$ be a SAT-instance whose clauses are either binary or spanning clauses containing a single negative literal as its last literal. There is a literal l contained in all non-binary clauses of $\mathcal{I}$.*

Proof. Consider the first variable x_j appearing in $\mathcal{I}$. Since a non-binary clause of $\mathcal{I}$ is a spanning clause (with a single negative literal as its last literal) of length at least 3, x_j appears only positively in such a clause, completing the proof. $\square$

Proposition 7. *Let $x_1, \ldots x_m$ be an ordered sequence of Boolean variables and $\mathcal{I}$ a set of clauses formed only of binary and spanning clauses. $\mathcal{I}$ is solvable in polynomial time by the algorithm of Figure 5.*

Proof. We must show that the algorithm of Figure 5 runs in polynomial time and that it returns the correct answer.

Let us first show that the algorithm always halts. This follows from the fact that *SpanSat* either reaches a return SAT/UNSAT statement (since $LBinSat$ halts by Proposition 5) or it recursively calls itself at line 16. Note that in this last case the number of variables as been decrease by at least one, hence *SpanSat* always halts.

Since total work is at most one recursion at each step plus polynomial work, *SpanSat* runs in polynomial time.

Now in order to check that *SpanSat* returns the correct answer, consider the following cases.

If *SpanSat* returns on line 6, the instance is empty, hence satisfiable. Similarly if it returns on line 7, the instance contains an empty clause, it is hence clearly unsatisfiable.

Now if *SpanSat* execute line 8, the satisfiability status of the instance is not modified. The new instance is hence equisatisfiable with the original one. If *SpanSat* returns on line 9, the instance is clearly unsatisfiable.

At that point *SpanSat* will tentatively set the first variable still appearing in $\mathcal{I}$, x_i, to 1 and then to 0.

At line 12, setting $\alpha(x_i) := TRUE$, will either satisfy a spanning clause or transform it into a clause having a unique negative literal as last literal. If on line 13 removing implication chains leads to a contradiction, there is no satisfying assignment with $x_i = 1$, accordingly line 14 restore the instance to its value at line 10 and check whether $x_i = 0$ can lead to a solution. Setting $\alpha(x_i) = FALSE$ will either satisfy a spanning clause or transform it into another spanning clause (on the ordered sequence of remaining variables). We are then left with an instance containing binary and spanning clauses, but of smaller size. We can now solve this instance by recursion on line 16.

If on line 13 removing implication chains leads to no contradiction, at line 17 $\mathcal{I}$ will contain only binary and spanning clauses with a unique negative literal as last literal. By Proposition 6 the hypothesis of Proposition 5 is fulfilled. Accordingly the instance is satisfiable and a satisfying assignment will be given by $LBinSat$. $\square$

We therefore have the following polynomial time algorithm for constraints satisfaction. First convert the constraint into clauses. Second, divide this instance by regrouping the clauses containing the same d together. Solve these instances in polynomial time, by the above result. If all these instances are solvable, take D to be the domain containing the introduced witnesses. Setting X_i's region to be the set of d's such that the literal $d \in X_i$ is set to true, will give us the desired spatial sequence. We therefore have the following result.

Theorem 2. *A set of constraints on the sequence of variables $X_1, \ldots, X_n$ using predicates $R_\subseteq$, $R_\cap$, $R_\supseteq$, $R_\|$ and their negations, can be solved in polynomial time on general domains.*

4.2.2. Axis-Parallel Domain

One may wonder if a similar method could be applied to constraint satisfaction in the axis-parallel domain case. In fact we show in this section that this is not the case. Using graph-theoretical methods, we show that constraint satisfaction on axis-parallel domains is in fact NP-complete. But before, let us first recall some useful facts from graph theory.

Given a family of geometric regions $\mathcal{F}$, a graph (V, E) is said to be an *intersection* graph of $\mathcal{F}$, if one can map elements $v \in V$ to subsets $f(v) \in \mathcal{F}$, is such a way as $\{v_1, v_2\} \in E$ if and only if $f(v_1) \cap f(v_2) \neq \emptyset$. A graph is said to be of *boxicity* N if it is the intersection graph of a family of N-dimensional axis-parallel boxes. For instance a graph of boxicity 1, also known as an *interval graph*, is the intersection graph of a family of intervals.

It has been shown by [35, 49] that deciding whether a graph is of boxicity N, for $N \geq 2$, is NP-complete. We will now show that checking N-boxicity can be polynomially reduced to a constraint satisfaction problem on an N-dimensional domain, showing that constraint satisfaction in the axis-parallel case and dimension at least 2, is NP-complete.

Theorem 3. *Constraint satisfaction in the axis-parallel case and dimension at least* 2*, is NP-complete.*

Proof. The problem is clearly in NP since checking that an assignment of the variables to axis-parallel regions satisfies the constraint can be done in polynomial time. This follows from the fact that each individual constraint can be checked in polynomial time by Propositions 2 and 3.

In order to show that the problem is NP-complete, we will polynomially reduce N-boxicity to a constraint satisfaction problem on an axis-parallel domain.

Take a graph (V, E), order its vertices in some arbitrary way and introduce for every $v \in V$ a region variable X_v, forming a sequence $X_{v_1}, \ldots, X_{v_n}$. For all $i < j$, add the constraint $X_{v_i} \cap X_{v_j} \neq \emptyset$ when $\{v_i, v_j\} \in E$ and the constraint $X_{v_i} \cap X_{v_j} = \emptyset$ when $\{v_i, v_j\} \notin E$. Clearly this set of constraints are satisfiable on a domain of dimension N if and only if (V, E) is of boxicity N. $\square$

Unfortunately this method cannot show NP-hardness for the 1-dimensional case, since [13] (see also [29]) have shown that 1-boxicity can be recognized in linear time. We therefore have to rely on a stronger graph-theoretical condition.

Let E_1 and E_2 be disjoint sets of edges on the same set of vertices V. A graph (V, E) such that $E_1 \subseteq E \subseteq E_1 \cup E_2$ is said to be a *sandwich graph* for (E_1, E_2). A sandwich graph (V, E) for (E_1, E_2) *must* hence contains all edges of E_1 but *may* also contains some of the edges of E_2.

The *interval graph sandwich problem* is the problem, given two graphs $G_1 = (V, E_1)$, $G_2 = (V, E_2)$ on the same set of vertices having disjoint edges sets (i.e. $E_1 \cap E_2 = \emptyset$), to test whether there is a sandwich graph for (E_1, E_2), which is an interval graph. This problem has been shown to be NP-complete by [25].

Theorem 4. *Constraint satisfaction in the axis-parallel case and dimension* 1*, is NP-complete.*

Proof. As before this problem is clearly in NP.

To show NP-completeness, we will reduce the interval graph sandwich problem to the satisfaction of constraints in dimension 1. Given the graphs (V, E_1) and (V, E_2), we introduce as before for every $v \in V$ a region X_v but this time we only add the constraints $X_{v_i} \cap X_{v_j} \neq \emptyset$ when $\{v_i, v_j\} \in E_1$. This will ensure that a solution to our CSP will give a graph (V, E) such that $E_1 \subseteq E$. In order to make sure that $E \subseteq E_1 \cup E_2$, we now add the constraints $X_{v_i} \cap X_{v_j} = \emptyset$ when $\{v_i, v_j\} \notin E_1 \cup E_2$. Testing for satisfiability on 1-dimensional domains being equivalent to solving the interval graph sandwich problem, we have shown the result. □

4.3. Model-checking

We now consider in this section the problem of checking whether a region of a spatial sequence satisfies some Visibility Logic formula.

We have already shown in section 4.1 that checking $R_{\subseteq}(A, B)$, $R_{\cap}(A, B)$, $R_{\supseteq}(A, B)$, $R_{\|}(A, B)$ on regions A, B of a spatial sequence can be done in polynomial time. We will now show that these results can be extended to check any Visibility Logic formula in polynomial time.

Theorem 5. *Model-checking a Visibility Logic formula on a spatial sequence can be done in time polynomial in the size of the sequence and formula.*

Proof. Consider a spatial sequence $\mathcal{R}$ and r a region of this sequence. We want to check whether $\mathcal{R}, r \models \varphi$, for φ a Visibility Logic formula. Call $\mathcal{R}, r \models \varphi$ an instance. Its size is the sum of the size of the spatial sequence $\mathcal{R}$ and formula φ.

In order to check $\mathcal{R}, r \models \varphi$, we will compute the truth-value of $\mathcal{R}, r' \models \psi$, for r' ranging over the regions of $\mathcal{R}$ and ψ ranging over all subformulas of φ (including φ).

As a first step, we start by building a table containing the truth-value of $R_{\subseteq}(r', r'')$, $R_{\cap}(r', r'')$, $R_{\supseteq}(r', r'')$ and $R_{\|}(r', r'')$ for all regions r, r' of $\mathcal{R}$. This can be done in time polynomial in the size of $\mathcal{R}$ using Propositions 2 and 3.

Secondly, we use the recursive nature of Visibility Logic semantics as presented in Section 3.2, to compute $\mathcal{R}, r' \models \psi$, for r' ranging over the regions of $\mathcal{R}$ and ψ ranging over all subformulas of φ. We use memoization and keep these values into a second table. This table has size $n \cdot |\varphi|$, where n is the number of regions in $\mathcal{R}$ and $|\varphi|$ the number of subformulas of φ. The size of this table is hence polynomial in the size of the instance. Searching an instance it this table can be done in constant time if it is a hash table and in linear time if it is simply a sequence. Either way, this will not influence the polynomial nature of the obtained bound. We will hence consider a constant time search in the remaining of this proof.

We now process the set of all $\mathcal{R}, r' \models \psi$ in increasing structural complexity of ψ. Therefore when some $\mathcal{R}, r' \models \psi$ is processed, then for any strict subformula ψ' of ψ and any region r'' of $\mathcal{R}$, all $\mathcal{R}, r'' \models \psi'$ have already been processed.

For ψ a propositional variable, the processing of a $\mathcal{R}, r' \models \psi$ can be done in constant time by checking the truth-value of ψ in region r'. For ψ the conjunction or disjunction of two formulas ψ', ψ', the processing of $\mathcal{R}, r' \models \psi$ can be done in constant time, by referring to the values of $\mathcal{R}, r' \models \psi'$ and $\mathcal{R}, r' \models \psi''$. Similarly for ψ the negation of a formula ψ', the processing of $\mathcal{R}, r' \models \psi$ can be done in constant time, by referring to the value of $\mathcal{R}, r' \models \psi'$.

For ψ the application of a modal connective to ψ', one first scans the spatial sequence for regions satisfying one of the binary relations $R_{\subseteq}$, $R_{\cap}$, $R_{\supseteq}$ and $R_{\|}$ according to the modal connector. This can be done in time linear in the spatial sequence's length n using our first table. For each of these regions r', one may have to check $\mathcal{R}, r' \models \psi'$, which can be done in constant time.

The processing of each $\mathcal{R}, r' \models \psi$ is hence done in polynomial time. Furthermore since there are only polynomial many such $\mathcal{R}, r' \models \psi$ the total work is polynomial. □

5. Applications

One encounters visibility considerations in many fields of computer science. We illustrate this fact by presenting in this section some formalizations using our relations and our logic. We present examples from two areas: image processing and computer networks. The reader is referred to [32] for how firewall filter verification can be realized with the help of a model-checker for Visibility Logic.

5.1. Image processing

Image processing plays an important role in many fields such as computational geometry, computer graphics, computer-aided design and robotics. Usually a digital image representation is built up from the representations of its individual regions. This allows analyzing the scene from characteristics and relative positions of the regions it contains.

For instance one of the most important computer graphics operations is the rendering of the global scene onto a two-dimensional frame; rendering being done is such a way as to display only regions that are visible from the viewpoint. This can be done in many ways, such as using a *depth order* [12], which is a linear ordering of regions, from foreground to background. The scene can then be rendered by simply drawing regions from background to foreground, avoiding the need to remove invisible parts, since they are simply overwritten.

A depth order can clearly be represented by a spatial sequence in which the regions appears from foreground to background. But usually a depth order is not unique. The only constraint is that if a region A obstructs a following region B (in our notation $R_{\cap}(A, B)$), A must appear before B in any depth order. We can express in Visibility Logic the fact that region r cannot be interchanged with its successor without violation depth order, in the following way.

$$\mathcal{R}, r \models \bigcirc_{\cap} \top$$

Axis-parallel regions have also attracted a considerable amount of attention in the computational geometry community. This is quite natural since axis-parallel regions appear in many applications such as electronic circuit masks or range queries in databases. There has been therefore many data structures and algorithms introduced to efficiently process axis-parallel regions.

For instance, rectangular range query [31] is the problem of listing, for some d-dimensional axis-parallel box, the points located inside it (Figure 6). This problem can be solved in $O(n \log n)$ time using Kd-trees [9] or range trees [10, 15, 21, 23, 45], where n is the number of points in a d-dimensional space. Furthermore, since region inclusion of axis-parallel regions ($A \subseteq B$) can be reduced to checking the inclusion of A's vertices in B, range query can easily be extended to searching which axis-parallel rectangles are completely contained in a d-dimensional axis-parallel query box.

Range query, both for points and axis-parallel rectangles can be represented in Visibility Logic in the following way. By ordering the given points (which are single element boxes) or axis-parallel rectangles, one obtains a spatial sequence. Adding the query box at the end of this spatial sequence and labeling it with a propositional variable q, true only on this last (query) box, the range query boils down to finding all regions r in this spatial sequence R satisfying the following formula.

$$R, r \models \Diamond_{\subseteq} q$$

A similar problem is the windowing query, which is to find axis-parallel line segments lying at least partially in an axis-parallel two-dimensional box (Figure 7). This problem can in fact be decomposed into two parts. First, to find the segments with at least one endpoint in the query window, which can be done by searching for the endpoints contained in the query window and is therefore a range query. The second part is to search for the segments with no endpoint in the query window. This case can be reduced to reporting all intervals containing a query point (see [11, Chapter 10]). This last operation can be done in time $O(n \log n)$, where n is the number of interval, using an interval tree [22, 39].

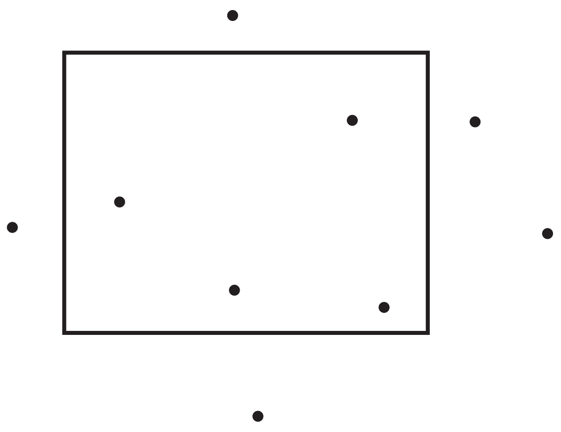
Figure 6: Range query

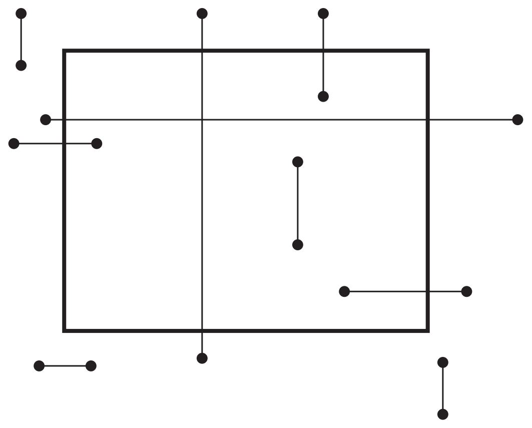
Figure 7: Windowing query

Rule	Source IP addr.	Dest. IP addr.	Decision
1	$*.*.*.*$	132.208.100.$*$	accept
2	130.$*.*.*$	132.208.$*.*$	deny
3	130.$*.*.*$	132.208.100.$*$	accept
4	$*.*.*.*$	$*.*.*.*$	deny

Figure 8: A schematic filter

Windowing query, can be represented in Visibility Logic by ordering the given line segments followed by the box (labeled by a propositional variable q) to obtain a spatial sequence. The windowing query turns out now to find all regions r in this spatial sequence R satisfying the following formula.

$$R, r \models \Diamond_{\cap} q$$

Axis-parallel regions are also used in computer-aided design and robotics, as a simple approximation for the space occupied by a region. Namely, a region's *bounding box* is the smallest axis-parallel box containing it. Since the bounding box is an over-approximation of the space occupied by a region, it is used in such area as computer-aided design [30], in order to rapidly determine regions that may collide. For instance in order to disassemble mechanical parts, one can ask for the parts in assembly order, which can be removed without first removing some other part.

Given a viewpoint on an assembly, we represent the orthogonal projections of the assembly's parts as a spatial sequence R, in assembly order. An assembly part is hence represented by the bounding box of its orthogonal projection onto the two-dimensional view frame. A part represented by r can be immediately removed, if no other part occludes it, which can be formalized by stating that no previous bounding box intersects r.

$$R, r \models \Box_{\cap}^{-1} \perp$$

5.2. *Firewall Filters*

In order to secure a corporate network or some subnetwork within it, network traffic is usually filtered according to such criteria as origin, destination, protocol and service. Typically, dedicated routers, called *firewalls*, are equipped with *filters* specifying which packets should be forwarded and which should be discarded.

A *filter* is a sequence of *rules* that are tried in order, up to the first matching one. A rule consists of a *condition*, which is a region of the packet's space (usually consisting of source/destination IP addresses, protocol, source/destination ports), and of a *decision* (usually accept/deny). Each packet hence goes through the rules in sequence up to the first matching condition, whose decision determines whether the packet is forwarded or discarded. In our setting a filter is simply formalized by its sequence of conditions, which are axis-parallel packet space regions and a unique propositional variable a representing the *accept* decision.

Figure 8 illustrates a schematic filter, where we consider only source and destination IP addresses represented in dotted decimal (four dot-separated numbers in the range $0 - 255$). To simplify presentation, we use a $*$ to represent the complete range $0 - 255$. In this figure the first rule expresses the fact that packets with any source IP address and destination in the range 132.208.100.$*$ are accepted. The second rule ensures that remaining packets with source address in the range 130.$*.*.*$ and destination in the range 132.208.$*.*$ will be denied. Note that the last rule ensures that all packets that match no previous rule will be denied.

Configuring a filter is a well-known error-prone task. Network management researchers have therefore introduced filter properties, called *anomalies*, which either reveal or hint to a possible misconfiguration.

For instance [2, 3] considered the following cases, involving a pair of rules.

A rule r_1 is *simply shadowed* if there is a rule r_2 preceding r_1 in the filter and such that all packets satisfying r_1's condition already satisfy r_2's. In such a case r_1 applies to no traffic and is therefore either misplaced or unneeded. This can be formalized as follows:

$$\mathcal{R}, r_1 \models \Diamond_{\supseteq}^{-1}\top$$

For instance rule 3 is simply shadowed by rule 1 in the filter of Figure 8.

Correlation happens when a later rule matches some packet already matched by r while having a different decision. In this case the filter is not necessarily misconfigured, but it could be useful to inform the network engineer that the second rule's decision will not apply to all packets satisfying its condition. We can formalize this property with the following formula.

$$\mathcal{R}, r \models (a \wedge \Diamond_{\cap}\neg a) \vee (\neg a \wedge \Diamond_{\cap} a)$$

For instance correlation happens for rules 1 and 2 in the filter of Figure 8.

Generalization happens when a later rule matches more than r, but has a different decision. While generalization has legitimate uses, such as rejecting packets from some host and then accepting traffic from all remaining machines on its subnet, it could be useful here also to inform the network engineer that the rule will not apply to all packets satisfying its condition. This property can be formalized as follows:

$$\mathcal{R}, r \models (a \wedge \Diamond_{\subseteq}\neg a) \vee (\neg a \wedge \Diamond_{\subseteq} a)$$

For instance rule 4 generalizes rule 1.

A rule is *simply redundant* if it is simply shadowed by a rule with the same decision. In this case, the rule can be removed without changing the packets that are accepted. This property can be formalized as follow.

$$\mathcal{R}, r_1 \models (a \wedge \Diamond_{\supseteq}^{-1} a) \vee (\neg a \wedge \Diamond_{\supseteq}^{-1}\neg a)$$

This is the case for rule 3 that is simply shadowed by rule 1 in the filter of Figure 8.

These anomalies were generalized by [4, 19] to consider interaction between more than two rules. For instance a rule r is *shadowed* if all packets satisfying the rule's condition already satisfy some previous rule's condition (different packets can now satisfy different rules' conditions).

This property can be formalized in Visibility Logic in the following way. If a rule r is the first rule matching some packet, then since there is necessarily a next rule matching this packet (according to the convention that the last rule matches the entire space), we have that r satisfies $\Diamond_{\|}\top$. Conversely, if r satisfies $\Diamond_{\|}\top$, then r is the first rule matching some packet. We therefore have that r is shadowed exactly when it matches no packet, which can be stated as follows.

$$\mathcal{R}, r \models \Box_{\|} \bot$$

Finally [27, 36] defined a rule to be *redundant* if removing it does not change the packets that are accepted. Here it is sufficient to state that every packet with r as first-matching rule has a second-matching rule with the same decision. This can be formalized as follows.

$$\mathcal{R}, r \models (a \wedge \Box_{\|} a) \vee (\neg a \wedge \Box_{\|}\neg a)$$

The network management community has proposed algorithms to check these anomalies. The algorithms presented in [2, 3] compare two regions for inclusion or intersection by simply comparing edges' endpoints. In order to consider properties relative to the complete regions' sequence, such as a region being included in the union of its predecessors, [4, 19] transforms the regions into a new sequence of disjoint regions. This method increases the total number of regions and an $O(d^n)$ upper bound on the running time is given, where d is the space's dimension, while n is the number of original regions.

Alternately, Binary Decision Diagrams have been considered in [51] to represent regions and test for the shadowing, generalization, correlation and redundancy anomalies. This approach has been evaluated experimentally; efficiently detecting anomalies in real filters such as checking a 800 rules filter in less than 3 seconds. A tree structure that represents a spatial decomposition of regions into non-overlapping axis-parallel regions is used in [50] in a prototype tool. Special decision tree data structures have also been introduced in [27, 36] to process sequences of regions, but with neither theoretical nor experimental evaluation.

In fact, since anomalies are properties definable in Visibility Logic, checking an anomaly reduces to model-checking, which can be done in polynomial time by the result of Section 4.3. Visibility Logic hence offers a uniform approach to anomaly verification, instead of relying on algorithms tailored to specific anomalies.

6. Conclusion

We considered properties of sequences of spatial regions, as seen from a viewpoint and introduced binary relations allowing to express properties of these sequences. Considering constraints satisfaction, we showed that it can be solved in polynomial time for general domain and is NP-complete for axis-parallel domain. We also introduced a modal logic on these relations, showing that model-checking on a finite sequence can be done in polynomial time; this both in the general and axis-parallel cases.

Visibility Logic allows us to express, in a uniform and succinct notation, a number of concepts ranging from range and window queries in computational geometry, to the verification of configuration anomalies in rule sets of network firewalls. In this latter case, we have even shown how our logical formulation immediately entails a polynomial algorithm for a problem whose only published solutions at this time are exponential.

7. Acknowledgments

We gratefully acknowledge the financial support of the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Fonds québécois de recherche sur la nature et les technologies (FQRNT). Special thanks also extend to Jean-Lou De Carufel for making us aware of results in computational geometry.

References

[1] M. Aiello, I. Pratt-Hartmann, J. van Benthem (Eds.), Handbook of Spatial Logics, Springer, 2007.
[2] E. Al-Shaer, H. Hamed, R. Boutaba, M. Hasan, Conflict classification and analysis of distributed firewall policies, IEEE Journal on Selected Areas in Communications 23 (2005) 2069–2084.
[3] E.S. Al-Shaer, H.H. Hamed, Discovery of policy anomalies in distributed firewalls, in: The 23rd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), pp. 2605–2616.
[4] J.G. Alfaro, N. Boulahia-Cuppens, F. Cuppens, Complete analysis of configuration rules to guarantee reliable network security policies, International Journal of Information Security 7 (2008) 103–122.
[5] J.F. Allen, Maintaining knowledge about temporal intervals, Communications of the ACM 26 (1983) 832–843.
[6] B. Aspvall, M.F. Plass, R.E. Tarjan, A linear-time algorithm for testing the truth of certain quantified boolean formulas, Information Processing Letters 8 (1979) 121–123.
[7] P. Balbiani, J.F. Condotta, L.F. Del Cerro, A new tractable subclass of the rectangle algebra, in: Proceedings of the 16th international joint conference on Artifical intelligence - Volume 1, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999, pp. 442–447.
[8] B. Bennett, Spatial reasoning with propositional logics, in: Principles of Knowledge Representation and Reasoning: Proceedings of the 4th International Conference (KR94), Morgan Kaufmann, 1994, pp. 51–62.
[9] J.L. Bentley, Multidimensional binary search trees used for associative searching, Communications of the ACM 18 (1975) 509–517.
[10] J.L. Bentley, Decomposable searching problems, Information Processing Letters 8 (1979) 244–251.
[11] M. Berg, O. Cheong, M. Kreveld, M. Overmars, Computational Geometry: Algorithms and Applications, Springer Verlag, 2008.
[12] M.d. Berg, M. Overmars, O. Schwarzkopf, Computing and verifying depth orders, SIAM Journal on Computing 23 (1994) 437–446.
[13] K.S. Booth, G.S. Lueker, Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms, Journal of Computer and System Sciences 13 (1976) 335–379.
[14] R. Casati, A.C. Varzi, Parts and Places: The Structures of Spatial Representation, MIT Press, 1999.
[15] B. Chazelle, Lower bounds for orthogonal range searching: I. The reporting case, Journal of the ACM 37 (1990) 200–212.

[16] B.L. Clarke, A calculus of individuals based on 'connection', Notre Dame Journal of Formal Logic 22 (1981) 204–218.

[17] B.L. Clarke, Individuals and points, Notre Dame Journal of Formal Logic 26 (1985) 61–75.

[18] A.G. Cohn, S.M. Hazarika, Qualitative spatial representation and reasoning: An overview, Fundamenta Informaticae 46 (2001) 1–29.

[19] F. Cuppens, N. Cuppens, J. Garcia-Alfaro, Misconfiguration management of network security components, in: Proceedings of the 7th International Symposium on System and Information Security (SSI 2005), ITA Corporate, Sao Paulo, Brazil, 2005, p. (electronic medium).

[20] C. Dornheim, Undecidability of plane polygonal mereotopology, in: Principles of Knowledge Representation and Reasoning: Proceedings of the 6th International Conference (KR-98), Morgan Kaufman, 1998, pp. 342–353.

[21] C. Duncan, M. Goodrich, Approximate geometric query structures, in: D.P. Mehta, S. Sahni (Eds.), Handbook Of Data Structures And Applications (Chapman & Hall/Crc Computer and Information Science Series.), Chapman & Hall/CRC, 2004. Chapter 26.

[22] H. Edelsbrunner, Dynamic data structures for orthogonal intersection queries, Technical Report Report F59, Inst. Information Process., Graz Univ. Technology, Austria, 1980.

[23] H. Edelsbrunner, L.J. Guibas, J. Stolfi, Optimal point location in a monotone subdivision, SIAM Journal on Computing 15 (1986) 317–340.

[24] D.M. Gabbay, A. Kurucz, F. Wolter, M. Zakharyaschev, Many-Dimensional Modal Logics - Theory and Applications, volume 148 of *Studies in Logic and the Foundations of Mathematics*, Elsevier, 2003.

[25] M.C. Golumbic, R. Shamir, Complexity and algorithms for reasoning about time: a graph-theoretic approach, Journal of the ACM 40 (1993) 1108–1133.

[26] N.M. Gotts, Using the RCC Formalism to Describe the Topology of Spherical Regions, Technical Report, Report 96.24, School of Computer Studies, University of Leeds, 1996.

[27] M.G. Gouda, A.X. Liu, Structured firewall design, Computer Networks 51 (2007) 1106–1120.

[28] P. Guha, A. Mukerjee, K.S. Venkatesh, OCS-14: You can get occluded in fourteen ways, in: T. Walsh (Ed.), Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence, AAAI Press/International Joint Conferences on Artificial Intelligence, Menlo Park, California, 2011, pp. 1665–1670.

[29] M. Habib, R. McConnell, C. Paul, L. Viennot, Lex-BFS and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing, Theoretical Computer Science 234 (2000) 59–84.

[30] H.J. Haverkort, M. de Berg, J. Gudmundsson, Box-trees for collision checking in industrial installations, Computational Geometry: Theory and Applications 28 (2004) 113–135.

[31] P. Houthuys, Box sort, a multidimensional binary sorting method for rectangular boxes, used for quick range searching, The Visual Computer 3 (1987) 236–249. 10.1007/BF01952830.

[32] B. Khorchani, S. Hallé, R. Villemaire, Firewall anomaly detection with a model checker for visibility logic, in: Proceedings of the IFIP/IEEE Network Operations and Management Symposium (NOMS 2012), IEEE Communications Society, 2012.

[33] H. Kleine Büning, T. Letterman, Propositional Logic: Deduction and Algorithms, Cambridge University Press, New York, NY, USA, 1999.

[34] B. Konev, R. Kontchakov, F. Wolter, M. Zakharyaschev, Dynamic topological logics over spaces with continuous functions, in: G. Governatori, I.M. Hodkinson, Y. Venema (Eds.), Advances in Modal Logic, College Publications, 2006, pp. 299–318.

[35] J. Kratochvíl, A special planar satisfiability problem and a consequence of its NP-completeness, Discrete Applied Mathematics 52 (1994) 233–252.

[36] A. Liu, M. Gouda, Complete redundancy detection in firewalls, in: Proceedings of the 19th Annual IFIP WG 11.3 Working Conference on Data and Applications Security, volume 3654 of *Lecture Notes in Computer Science*, Springer, 2005.

[37] Z. Manna, A. Pnueli, The Temporal Logic of Reactive and Concurrent Systems:Specification, Springer-Verlag, 1992.

[38] Z. Manna, A. Pnueli, Temporal Verification of Reactive Systems: Safety, Springer-Verlag, 1995.

[39] E. McCreight, Efficient algorithms for enumerating intersecting intervals and rectangles, Technical Report Report CSL 80-9, XEROX PARC, 1980.

[40] D.A. Randell, Z. Cui, A.G. Cohn, A spatial logic based on regions and connection, in: KR'92: 3rd International Conference on Knowledge Representation and Reasoning, Morgan Kaufmann, 1992, pp. 165–176.

[41] J. Renz, B. Nebel, On the complexity of qualitative spatial reasoning: a maximal tractable fragment of the region connection calculus, Artificial Intelligence 108 (1999) 69–123.

[42] R.E. Tarjan, Depth-first search and linear graph algorithms, SIAM Journal on Computing 1 (1972) 146–160.

[43] L. Vieu, Spatial Representation and Reasoning in Artificial Intelligence, in: O. Stock (Ed.), Spatial and Temporal Reasoning, Kluwer, 1997, pp. 3–41.

[44] R. Villemaire, S. Hallé, Strong temporal, weak spatial logic for rule based filters, in: TIME '09: Proceedings of the 2009 16th International Symposium on Temporal Representation and Reasoning, IEEE Computer Society, Washington, DC, USA, 2009, pp. 115–121.

[45] D.E. Willard, The Super B-Tree Algorithm, Technical Report TR-03-79, Aiken Computer Laboratory, Harvard University, 1979.

[46] F. Wolter, M. Zakharyaschev, Spatial reasoning in RCC-8 with boolean region terms, in: W. Horn (Ed.), Proceedings of the fourteenth European Conference on Artificial Intelligence, ECAI 2000, Berlin, Germany, IOS Press, 2000, pp. 244–248.

[47] F. Wolter, M. Zakharyaschev, Spatio-temporal representation and reasoning based on RCC-8, in: Proceedings of the seventh Conference on Principles of Knowledge Representation and Reasoning, KR2000, Morgan Kaufmann, 2000, pp. 3–14.

[48] F. Wolter, M. Zakharyaschev, Qualitative spatio-temporal representation and reasoning: a computational perspective, in: Exploring Artifitial Intelligence in the New Millenium, Morgan Kaufmann, 2002, pp. 175–216.

[49] M. Yannakakis, The complexity of the partial order dimension problem, SIAM Journal on Algebraic and Discrete Methods 3 (1982) 351–358.

[50] Y. Yin, R. Bhuvaneswaran, Y. Katayama, N. Takahashi, Analysis methods of firewall policies by using spatial relationships

between filters, in: International Conference on Signal Processing, Communications and Networking (ICSCN ’07), pp. 348–354.
[51] L. Yuan, J. Mai, Z. Su, H. Chen, C.N. Chuah, P. Mohapatra, FIREMAN: A toolkit for firewall modeling and analysis, in: IEEE Symposium on Security and Privacy, IEEE Computer Society, Los Alamitos, CA, USA, 2006, pp. 199–213.