The Impact of Qualification on the Application of Qualitative Spatial and Temporal Reasoning Calculi

Carl Schultz[1] and Robert Amor[2] and Hans W. Guesgen[3]

[1] SFB/TR 8 Spatial Cognition, The University of Bremen, Germany
[2] The University of Auckland, New Zealand
[3] Massey University, New Zealand

Abstract. Ever since Allen introduced his qualitative interval algebra in 1983, the area of qualitative spatial and temporal reasoning (QSTR) has been motivated by potential application areas that require human-oriented, commonsense reasoning. Despite this, it is well recognised in the community that there are relatively few commercial applications that heavily employ QSTR calculi. In this paper we directly address this issue by establishing a theoretical foundation for describing, developing and analysing QSTR based applications. We present an analysis of QSTR calculus qualification and investigate the impact that qualification has on a QSTR application's reasoning properties such as completeness and soundness. Our definition of QSTR applications also provides software developers with a basic template to begin creating their own applications. Concrete examples of existing QSTR applications are used to demonstrate and motivate this research.

1 Introduction

Qualitative spatial and temporal reasoning (QSTR) calculi represent and reason about coarse, intuitive relations between objects. The most prominent QSTR calculus is Allen's Interval Algebra (IA) [1]; Allen defines thirteen jointly exhaustive, pairwise disjoint relations that can hold between a pair of temporal intervals: before, meets, overlaps, starts, during, finishes, after, met by, overlapped by, started by, contains, finished by, and equals. Allen's seminal contribution was to frame the problem of determining a minimally consistent qualitative description of object relations as a constraint satisfaction problem, and proposed a modified path-consistency algorithm that performs composition using a reference look-up table.[4] Allen's approach to qualitative temporal reasoning has motivated the development of a large number of QSTR calculi [2]. For example, Region Connection Calculus 8 (RCC8) defines eight topological relations that can hold between pairs of arbitrary regular regions [3]: disconnected (DC), externally connected (EC), partially overlaps (PO), tangential proper part (TPP), non-tangential proper part (NTPP), tangentially contains (TPPi), non-tangentially

[4] Composition of two relations $R1$ and $R2$ produces the relation $R3$ such that for all x, y, z if $R1(x, y)$ and $R2(y, z)$ then $R3(x, z)$.

contains (NTPPi), and equals (EQ). As with IA, a reference table is used to implement the composition operator.

A qualitative representation of object relations is formulated as a constraint satisfaction problem in the following way [1]. A constraint network consists of a finite set of vertices (representing objects in the world) and directed edges between vertices (representing relations between objects). Each edge is a variable that contains a set of qualitative relations that can possibly hold between two objects. A network is non-atomic if at least one edge contains more than one possible relation, and a network is atomic if all edges contain exactly one relation. Allen's algorithm attempts to eliminate relations from each edge which are not consistent based on the composition of relations [1].

Most QSTR calculi come with (sometimes implicit) domains of interpretation [4], for example, a constraint network using IA relations is interpreted as a linear ordering W together with a subset U of the intervals $(w1, w2)$, $w1 < w2$ on W. The process of mapping a configuration in the domain of interpretation to a qualitative constraint network is *qualification*, which we denote as the relation map q. A network is consistent if, for each relation $R(x, y)$ there is some consistent instantiation of all objects in the domain of interpretation that also satisfies R. A network is path-consistent if, for all triples of variables x, y, z, any consistent instantiation of x, z can be extended to some consistent instantiation of y. A well recognised concept in QSTR is weak composition (or extensionality [5, 6]). It was noticed that Allen's algorithm applied to different calculi does not always give path-consistency but algebraic closure, i.e. $\forall x, y, z \cdot R(x, y) \wedge S(y, z) \rightarrow T_1(x, z) \vee \ldots \vee T_n(x, z)$. In particular, the consequent is a necessary but not a sufficient condition as required by path-consistency. Therefore algebraic closure is weaker than path-consistency and some impossible relations may not be eliminated from the constraint network.

The development of QSTR calculi is very often motivated by potential application areas that require more coarse, intuitive reasoning. For example, Egenhofer developed qualitative approaches to facilitate querying in GIS [13], and Wolter et al. developed vessel navigation application using qualitative orientation [12]. However, it is well recognised in the community that there is an absence of commerical applications, developed by application domain experts, that heavily employ QSTR calculi [9, 10]. Relatively little research in QSTR has focused directly on the issues faced by software developers that are interested in applying QSTR calculi. This paper addresses the needs of application developers by establishing a formal definition of QSTR applications and analysing the impact of qualification on the properties of QSTR application reasoning. Section 2 provides a formal definition of QSTR applications. Section 3 presents an approach to analysing QSTR calculi with respect to application requirements. Section 4 analyses completeness and soundness of QSTR applications based on the given definition. Section 5 presents the conclusions of the paper.

2 A General Definition of QSTR Applications

This section presents a general definition of QSTR applications that is simple and yet sufficient for modelling all existing QSTR applications that the authors are aware of, regardless of the application domain. The benefits of formally defining QSTR applications are firstly that the general properties of applications can be analysed in detail and secondly that it provides software developers with a basic template to begin creating their own applications. To demonstrate this a number of examples of existing QSTR applications are given using this formulation.

One approach that has been used to apply QSTR calculi is to treat network inconsistency as a metaphor for some undesirable condition which is specific to the application domain. For example, Nokel [11] (page 46, Figure 23) reasons about three valves attached to a tank where exactly one valve must be open at any time; an inconsistent network means invalid valve behaviour. There are three main problems with the metaphor approach. Firstly, by mixing application-specific criteria with general laws of spatial arrangements the meaning of consistency becomes overloaded and heavily dependent on the application. Secondly the metaphors can be awkward and difficult to implement without an advanced understanding of compositional reasoning. Thirdly the metaphor approach is restricted to constraining triples of binary relations which greatly reduces the expressiveness of possible QSTR applications.

We will now present our alternative definition of QSTR applications. We define QSTR applications as having either (a) numerical runtime input which is qualified, (b) qualitative runtime input, or (c) qualitative rules that determine how the application should respond to input. Using this basic formulation we characterise the general runtime behaviour as follows: (1) receive input (2) (optional) do qualification (3) construct constraint network (4) (optional) do consistency check (5) execute application rules (6) return output.[5] Formally a QSTR application is a function that maps input symbols to sets of output symbols (i.e. inferred expressions). The input is a set of expressions that represent premises (or facts) about the world description (or a sequence of n world descriptions); e.g., the input expressions can use numerical relations (e.g. *kitchen.x=550* and *kitchen.y=100*) or qualitative relations (e.g. *near(kitchen, livingRoom)*). The output is a set of expressions that is some relevant subset of all inferred facts (as defined by the developer); e.g., output symbol α may correspond to the expression $\exists x \in U \cdot bathroom(x) \wedge near(x, kitchen)$.

Definition 1. Let T_i be a set of expressions in some input language, let O be a set of output symbols representing logical expressions, and let n be a natural number. A QSTR application A is a function $A(T_1, \ldots, T_n) \subseteq 2^O$.

The rules used to infer new facts based on the input premises are first-order constraints. If a constraint is not satisfied then a new fact is inferred in order to satisfy the constraint. QSTR calculi typically represent uncertainty by

[5] In this paper we focus on purely qualitative applications. Hybrid applications integrate both qualitative and numerical rules.

maintaining disjunctions of relations that can *possibly* hold between two objects. Because QSTR applications infer new facts using QSTR calculi relations that *possibly* hold, then the application inferences produced must also be interpreted by the user as *possibly* holding. This formulation will now be used to define a number of existing QSTR applications.

SailAway [12] Input is a configuration of vessels consistent with a qualitative orientation calculus T^{OPRA}. Rules T^X are a formalisation of maritime right-of-way rules that use simple custom qualitative relations such as *collisionAtRear* and vessel types such as *motorVessel*. Output is a (set of) sequences of $OPRA$ constraint networks that are consistent with T^X (i.e. simulates future vessel states and eliminates states that cause collisions).

GIS QueryBySketch [6] [13] Input is a set of bitmap representations of objects parsed into vector representations. Qualification creates a constraint network with 9-Intersection relations, custom detailed topological relations, and cardinal directions (salient numerical information is also maintained). Output (of the qualitative module) is simply the constraint network with optional relaxation information such as neighbouring relations which is used by a query processor.

Tank-valves [11] Input is a configuration consistent with T^{IA}. Output symbol γ indicates abnormal valve operation. Rules for generating γ are expressions where x,y,z are the intervals when valves are open: (1) $overlaps(x, y) \vee starts(x, y) \vee during(x, y) \vee finishes(x, y)$ (2) $before(x, z) \wedge \neg\exists y \cdot meets(y, z)$.

Lighting Design [14] Input is a set of numerical and qualitative expressions about spatial objects consistent with a qualitative orientation calculus T^{BA} (block algebra). Rules formalise lighting principles (e.g. *brightAmbientIllumination*) and higher level principles about subjective impressions (e.g. *spaciousness*). Output is the subset of inferences that specify the subjective impressions of rooms.

3 Selecting QSTR Calculi

A critical role of the QSTR application developer is deciding which collection of QSTR calculi, if any, should be used. Important factors include complexity of reasoning and ontological requirements. However, even when a calculus is ontologically appropriate, the relations may be too coarse on their own to accurately model a particular application concept. We will now present an analysis of qualification to greatly assist in both the selection of QSTR calculi and the development of QSTR application rules. Qualification is critically important because when rules are defined using particular calculi the application is not able to distinguish between certain geometric configurations.[7] The power of this analysis is that it integrates major areas of QSTR research [5, 6, 15, 8] to give the developer direct insights into calculi under different key input conditions.

[6] This is a hybrid application; we only emphasise the qualitative components.

[7] Qualification depends on the properties of QSTR calculi and is completely independent of the QSTR application formulation in the previous section.

Determining whether a given constraint network is consistent is in general an intractable task [8]. Importantly, not all applications actually need to check the consistency of the input constraint networks as in many domains the input is guaranteed to be consistent (e.g. given by CAD tools). We will therefore firstly consider the case where input constraint networks are known to be consistent a-priori.

3.1 Qualification When Input is a Consistent Atomic Network

We will firstly consider the case of atomic constraint networks that are known to be consistent a-priori.[8] Given two objects (for binary calculi) the pertinent information about distinguishing between geometric configurations is given directly by the qualification operator. For example, in IA *before(x,y)* is defined as $x^+ < y^-$ (where t^- and t^+ are the start and end points of interval t respectively), and in RCC8 *EC(x,y)* is defined as $x \cap y \neq \emptyset \wedge i(x) \cap i(y) = \emptyset$ (where $i(r)$ is the interior of region r). The developer simply needs to review the qualification operator (in practice, a software tool would be used to conduct this analysis). Determining qualification of three or more objects (for binary relations) is simply the conjunction of the qualifications of each individual pair of objects. Thus the set of geometric instances that satisfy $q(R_1(x_1, x_2))$ is a superset of $q(R_1(x_1, x_2) \wedge ... \wedge R_n(x_{2n-1}, x_{2n}))$. Importantly, if $R_1(x, y)$ in isolation is too coarse then by combining relations from different calculi the developer can design rules that more closely approximate the necessary distinctions [15].

If the developer knows two geometric configurations that need to be distinguished then they can easily determine whether some combination of relations is adequate.[9] A calculus can distinguish two geometric configurations that are described as a set T_i^c of numerical expressions if $q(T_1^c) \not\leftrightarrow q(T_2^c)$. Furthermore, the precise geometric configurations that are numerically distinct but qualitatively indistinguishable from a given geometric configuration T^c are given by the expression $\neg q(T^c) \wedge q^{-1}(q(T^c))$ (where q^{-1} gives the numerical expression that corresponds to the given qualitative expression, i.e. the inverse of qualification).

For example, a lighting designer wants to specify patches of light on walls where two spotlights are directed. Their specific criteria are that the entire wall is covered in light (with some light spillage) while no light patches overlap: (1) $Wall \subset L_1 \cup L_2$ (2) $\emptyset = i(L_1) \cap i(L_2)$. This is qualified in RCC as: (1) $EC(L_1, L_2)$ (2) $PO(Wall, L_1)$ (3) $PO(Wall, L_2)$. The unwanted geometric configurations are: $\neg q(T^c) \wedge q^{-1}(q(T^c)) = (Wall \not\subset L_1 \cup L_2) \wedge i(Wall) \cap i(L_1) \neq \emptyset \wedge i(Wall) \cap i(L_2) \neq \emptyset \wedge i(L_1) \cap i(L_2) = \emptyset$. The developer responds by adding a new relation $covered(w) \equiv \forall r \cdot C(w, r) \rightarrow \exists l \cdot light(l) \wedge C(r, l)$. However, *covered* cannot be calculated because it refers to arbitrary regions r connected to the

[8] The consistency check is not performed thus avoiding the difficulties that some QSTR calculi have in determining the consistency of atomic networks.

[9] Qualitative calculi are jointly exhaustive and so every operator used to describe a geometric configuration corresponds to one, or the disjunction of more than one, qualitative relation.

wall w that have not been stored in the constraint network. Thus, the developer includes another custom relation (computed during qualification before the constraint network is constructed): $covers(w, r) \equiv r = \bigcup\{r_i | C(w, r_i)\}$. Even though ontologically RCC is very useful this additional analysis makes it clear that RCC in isolation is not sufficient and helps to guide the developer in creating custom relations.

3.2 Qualification When Input is a Consistent Non-Atomic Network

We now consider the case of atomic constraint networks that are known to be consistent a-priori for which some ambiguous information is added making the network non-atomic (for example, the world is only partially observable). Due to weak composition some QSTR calculi cannot always determine when a relation is impossible to instantiate i.e. the constraint network is consistent but not path-consistent [6]. Thus the application may erroneously make inferences based on impossible relations; clearly this is critical information for the developer. The explicit non-extensional composition triads given by the QSTR community (e.g. [6]) can provide the developer with meaningful information about the practical limitations of the calculus and the behaviour of applications built using those calculi.

We will use the following example to demonstrate that, due to weak composition, a calculus cannot always detect impossible relations. An art director of The Gallery of the Accademia di Belle Arti in Florence wants to temporarily exhibit a notable sculpture. The permanent gallery centrepiece is Michelangelo's David and the director wants to create a natural flow from David onto the temporary exhibit to offer a unique, exciting perspective. However, the new exhibit should not be immediately adjacent in case it distracts from the initial impact of David. There is some ambiguity in the design as the director has not yet decided where to place the temporary exhibit. The developer encodes the following rules: (1) $adjacent(x, y) \equiv EC(x, y)$ (2) $accessible(x, y) \equiv \exists w \cdot walkpath(w) \wedge EC(w, x) \wedge EC(w, y)$ (3) $interference(x) \equiv \exists y \cdot exhibit(y) \wedge adjacent(x, y)$ (4) $surrounds(x, y) \equiv EC(x, y) \wedge convexHull(h, x) \wedge PP(y, h)$ (PP is *proper part*). The floor plans are qualified as follows: (1) $walkpath(W1)$ (2) $surrounds(W1, David)$. The new exhibit will be placed in the same space as David to get the flow on effect: $surrounds(W1, NewExhibit)$. The question is whether this RCC representation is adequate. Notice that in this context *surrounds* implies that one region fills the hole of another region; this is a well known case of non-extensional reasoning in RCC. Thus, the application alerts the director to the possibility that the subjective impression of David is interfered with by the new exhibit although this is impossible as David is *surrounded* by the walkpath $W1$. Without having a detailed understanding of RCC this can be puzzling and frustrating to a user. The problem is that: $EC(David, WP1) \wedge EC(NewExhibit, WP1)$, so RCC8 erroneously infers that $EC(David, NewExhibit)$ is possible giving $adjacent(David, NewExhibit)$ and $interference(David)$. Indeed, any non-extensional triads pose a possible trap for the user and developer. After reviewing these triads (with the assistance of

QSTR application development tools) the developer can determine that other relations in addition to RCC8 relations are required to formalise *surrounds*.

3.3 Qualification When Input is an Inconsistent Network

In this section we analyse the case where the constraint network presented to the application is inconsistent in a way that is undetectable to the QSTR calculus. This covers a large number of real world application situations where there are conflicting sources of information about a world description. For example, a robot's sensory inputs may be in conflict, or eye-witness reports of the scene of a crime may disagree. Determining whether a constraint network is inconsistent is an intractable task, and thus there is the possibility that the application will process an inconsistent network without being notified by the QSTR calculi that the network is indeed inconsistent. It is highly important that the software developer knows how the application will behave if an inconsistent constraint network is undetected by the QSTR calculi, particularly for safety-critical applications.

In the previous two sections it was shown that the developer can determine exactly which world descriptions are indistinguishable, and thus they can predict the exact erroneous inferences that will be made. However, when networks are potentially inconsistent developers can no longer predict the exact erroneous inferences.[10]

Proposition 2. *Any QSTR application inference can be made when a network is undetectably inconsistent.*

Proof. Let N be an inconsistent network that is determined to be consistent by some calculus Q. Let e be an expression (constraint) that causes application A to produce output α. Let N' be some network that satisfies e and for which its vertex set is disjoint with the vertex set of N (no objects in common), then $N'' = N' \cup N$ is undetectably inconsistent by Q and A will erroneously produce α. □

This is a highly relevant property of QSTR calculi and developers need to know about the types of networks in which reasoning is potentially erroneous. The developer can then determine whether the risk of faulty reasoning under those specific circumstances is acceptable or unacceptable.

Firstly we consider atomic networks. For some QSTR calculi (but not all) algebraic closure is sufficient for determining consistency in atomic networks, including IA and RCC8. However, Renz and Ligozat [5] have proven that calculi that do not have a property called *closure under constraints* are not always capable of detecting an inconsistent atomic network. This is precisely the information that a developer needs to determine when an application may potentially produce incorrect inferences. For example, an art director has over constrained

[10] This means that the software developer cannot identify any vulnerable rule. It is important to stress that this is a property of QSTR calculi and not the application formulation in this paper.

the qualitative location of a light source that they are trying to place in their gallery. However, due to the limitations of their particular chosen QSTR calculi the fact that the atomic network is inconsistent has remained undetected. Based on the location of the light, a number of faulty inferences are produced including a design warning that the light source is visible as people enter the room and thus distracts from the impact of the other exhibits. This causes the director to needlessly spend considerable effort and time moving the exhibits around the room to fine-tune the design when in fact the design is physically unrealisable.

Determining whether a non-atomic network is consistent is an NP-hard problem for all QSTR calculi [8]. However, some of the most prominent advances in QSTR research have been the identification of maximal tractable subsets of a given calculus [8, 7]. The non-atomic networks for which an application may produce erroneous inferences is precisely the intractable subsets of the calculus, and it is therefore the information about intractable subsets that is critical to a developer. This concludes the analysis of QSTR calculi qualification and the possibility of an application producing erroneous inferences due to the qualification and consistency-checking limitations of QSTR calculi.

4 Properties of QSTR Applications

We will now use the qualification and consistency-checking properties of QSTR calculi presented in the previous sections to determine application completeness and soundness. A QSTR application is *complete* if the set of application inferences is always an improper superset of the set of correct inferences.

Proposition 3. *QSTR applications are complete.*

Proof. QSTR calculi are sound and therefore no correctly *possible* relations are eliminated. Thus, the constraint network contains the set of actual relations. QSTR applications make inferences based on all constraint network relations. Thus applications make all inferences that use the set of actual relations. □

Completeness is a very useful property of QSTR applications. It provides the user with some certainty in the context of intractable reasoning problems; they can be guaranteed that any inferences that the application did not make surely do not hold. A QSTR application is *sound* if the set of application inferences is always an improper subset of the set of correct inferences.

Proposition 4. *QSTR applications are not sound.*

Proof. Follows from the qualification analysis in Section 3. □

QSTR applications are not sound as a result of the properties of the underlying QSTR calculi. Thus, not all QSTR application inferences necessarily actually hold.[11] We will now show that in fact completeness only holds for inferences that

[11] This mirrors the properties of Allen's algebraic closure reasoning algorithm which is sound but not complete, thus leaving a set of *possible* relations which is a superset of the real set of *possible* relations.

require the constraint network to be consistent. Some applications may need to produce particular outputs (i.e. inferences) only when the network is found to be *inconsistent*. For example, when a robot detects an inconsistent network then it knows that something is wrong with its sensors and may need to execute a calibration routine to try to correct the sensor problem.

Proposition 5. *QSTR applications are neither complete nor sound for inferences on inconsistent networks.*

Proof. First, non-soundness by example. Let an application produce α if the network is both inconsistent and satisfies $EC(x, y) \wedge EC(y, z) \wedge EC(x, z)$. Let N be a non-atomic network that has been correctly identified as inconsistent by the application, which also contains $EC(A, B)$ and $EC(B, C)$ where region B completely fills a hole in A. RCC8 composition will erroneously infer that $EC(A, C)$ is a possible relation (i.e. this is an additional inconsistency that was *not* detected by the application) and the application will erroneously produce α. Second, non-completeness. Let N be an inconsistent non-atomic RCC8 network such that algebraic closure cannot detect the inconsistency. Then no inferences that hold in N when N is inconsistent will be produced. □

Importantly, soundness of QSTR application inferences on an inconsistent network obeys exactly the same principles as soundness of consistent network inferences. That is, if a network is found to be inconsistent then we can guarantee that the network is actually inconsistent, and thus the cases where the application inferences will be incorrect are exactly the same cases presented in Section 3. Completeness of inferences on inconsistent networks depends on the QSTR calculus correctly detecting when a network is inconsistent as discussed in Section 3.3.

5 Conclusions

We have presented research that supports the development of applications that incorporate QSTR calculi. We define QSTR applications as functions that accept spatial and temporal information, execute domain specific rules, and then return relevant inferences as the application output. This definition is capable of expressing all existing QSTR applications that the authors are aware of, and four examples of existing QSTR applications were given. We analysed qualification by considering four distinct cases based on whether the constraint network is atomic or non-atomic, and whether the network is known to be consistent a-priori. This analysis integrates major areas of research from the QSTR community such as tractable subsets and weak-composition and identifies how developers can use this research to assist in selecting appropriate calculi and developing domain specific rules. The qualification results were then used to determine some basic properties of QSTR application reasoning. We showed that when QSTR applications make inferences using consistent networks they are complete but not sound, and when applications make inferences (intentionally) using inconsistent

networks they are neither complete nor sound. An important area of future research is the human-computer interaction issue of enabling developers to analyse qualification from application rules in a natural way (e.g. using graphical languages).

References

1. J. F. Allen, Maintaining knowledge about temporal intervals, Communications of the ACM, 26, 832–843, (1983).
2. A. G. Cohn and J. Renz, Qualitative spatial reasoning, in Handbook of Knowledge Representation, eds., Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter, Elsevier, (2007).
3. D.A. Randell, Z. Cui, and A.G. Cohn, A spatial logic based on regions and connection, in Proc. 3rd Int. Conf. on Knowledge Representation and Reasoning, pp. 165–176, San Mateo, (1992). Morgan Kaufmann.
4. G. Ligozat, D. Mitra, and J.-F. Condotta, Spatial and temporal reasoning: beyond allens calculus, AI Communications, 17(4), 223–233, (2004).
5. J. Renz and G. Ligozat, Weak composition for qualitative spatial and temporal reasoning, in CP, ed., Peter van Beek, volume 3709 of Lecture Notes in Computer Science, pp. 534–548. Springer, (2005).
6. S. Li and M. Ying, Region connection calculus: Its models and composition table, Artif. Intell., 145(1-2), 121–146, (2003).
7. J. Renz, Maximal tractable fragments of the region connection calculus: A complete analysis, in IJCAI, ed., Thomas Dean, pp. 448–455. Morgan Kaufmann, (1999).
8. B. Nebel and H.-J. Burckert, Reasoning about temporal relations: A maximal tractable subclass of allens interval algebra, J. ACM, 42(1), 4366, (1995).
9. J. O. Wallgrun, L. Frommberger, D. Wolter, F. Dylla, and C. Freksa, Qualitative Spatial Representation and Reasoning in the SparQ-Toolbox. Spatial Cognition, 4387 LNCS, 39–58, Springer-Verlag Berlin Heidelberg, (2007).
10. T. Hahmann and M. Gruninger, Detecting physical defects: A practical 2d-study of cracks and holes, in AAAI Spring Symposium on Benchmarking of Qualitative Spatial and Temporal Reasoning Systems, Palo Alto, Technical Report SS-09-02, pp. 11–16. AAAI Press, (2009).
11. K. Nokel, Temporally Distributed Symptoms in Technical Diagnosis, volume 517 of Lecture Notes in Computer Science, Springer, 1991.
12. D. Wolter, F. Dylla, S. Wolfl, J. O. Wallgrun, L. Frommberger, B. Nebel, and C. Freksa, Sailaway: Spatial cognition in sea navigation, in Advances in Artificial Intelligence, 31st Annual German Conference on AI, vol. 22, no. 1. Kaiserslautern, Germany: Springer, September 2008, pp. 28–30.
13. M. J. Egenhofer: Query Processing in Spatial-Query-by-Sketch. J. Vis. Lang. Comput. 8(4): 403–424 (1997).
14. C. P. L. Schultz, R. Amor, B. Lobb, and H. W. Guesgen, Qualitative design support for engineering and architecture, Advanced Engineering Informatics, vol. 23, pp. 68–80, (2009).
15. S. Wolfl and M. Westphal, On combinations of binary qualitative constraint calculi, in 21st International Joint Conference on Artificial Intelligence, C. Boutilier, Ed., Pasadena, California, USA, July, pp. 967–973 (2009).

This article was processed using the LaTeX macro package with LLNCS style