Qualitative Spatial Representation and Reasoning in the SparQ-Toolbox

Jan Oliver Wallgrün, Lutz Frommberger, Diedrich Wolter, Frank Dylla, and Christian Freksa

SFB/TR 8 Spatial Cognition
Universität Bremen
Bibliothekstr. 1, 28359 Bremen, Germany
{wallgruen,lutz,dwolter,dylla,freksa}@sfbtr8.uni-bremen.de

Abstract. A multitude of calculi for qualitative spatial reasoning (QSR) have been proposed during the last two decades. The number of practical applications that make use of QSR techniques is, however, comparatively small. One reason for this may be seen in the difficulty for people from outside the field to incorporate the required reasoning techniques into their software. Sometimes, proposed calculi are only partially specified and implementations are rarely available. With the SparQ toolbox presented in this text, we seek to improve this situation by making common calculi and standard reasoning techniques accessible in a way that allows for easy integration into applications. We hope to turn this into a community effort and encourage researchers to incorporate their calculi into SparQ. This text is intended to present SparQ to potential users and contributors and to provide an overview on its features and utilization.

1 Introduction

Qualitative spatial reasoning (QSR) is an established field of research pursued by investigators from many disciplines including geography, philosophy, computer science, and AI [1]. The general goal is to model commonsense knowledge and reasoning about space as efficient representation and reasoning mechanisms that are still expressive enough to solve a given task. Qualitative spatial representation techniques are especially suited for applications that involve interaction with humans as they provide an interface based on human spatial concepts.

Following the approach taken in Allen's seminal paper on qualitative temporal reasoning [2], QSR is typically realized in form of calculi over sets of spatial relations (like 'left-of' or 'north-of'). These are called *qualitative spatial calculi*. A multitude of spatial calculi has been proposed during the last two decades, focusing on different aspects of space (mereotopology, orientation, distance, etc.) and dealing with different kinds of objects (points, line segments, extended objects, etc.). Two main research directions in QSR are mereotopological reasoning about regions [3,4,5] and reasoning about positional information (distance and orientation) of point objects [6,7,8,9,10,11,12] or line segments [13,14,15]. In

T. Barkowsky et al. (Eds.): Spatial Cognition V, LNAI 4387, pp. 39–58, 2007.

addition, some approaches are concerned with direction relations between extended objects [16,17] or combine different aspects of space [18,19].

Despite this large variety of qualitative spatial calculi, the amount of applications employing qualitative spatial reasoning techniques is comparatively small. We believe that one important factor for this is the following: Choosing the right calculus for a particular application is a challenging task, especially for people not familiar with QSR. Calculi are often only partially specified and usually no implementation is made available—if the calculus is implemented at all and not only investigated theoretically. As a result, it is not possible to "quickly" evaluate how different calculi perform in practice. Even if an application developer has decided on a particular calculus, he has to invest serious efforts to include the calculus and required reasoning techniques into the application. For many calculi this is a time-consuming and error-prone process (e. g. involving writing down large composition tables, which are often not even completely specified in the literature). We think that researchers involved in the investigation of QSR will also benefit from reference implementations of calculi that are available in a coherent framework. Tasks like comparing different calculi with respect to expressiveness or average computational properties in a certain context would clearly be simplified.

To provide a platform for making the calculi and reasoning techniques developed in the QSR community available, we have started the development of a qualitative spatial reasoning toolbox called *SparQ*[1]. The toolbox supports binary and ternary spatial calculi. SparQ aims at supporting the most common tasks—qualification, computing with relations, constraint-based reasoning (cp. Section 3)—for an extensible set of spatial calculi. Our focus is on providing an implementation of QSR techniques that is tailored towards the needs of application developers. A similar approach has recently been reported in [20] where calculi and reasoning techniques are provided in form of a programming library and focuses on algebraic and constraint-based reasoning. SparQ, on the other hand, is a application program that can be used directly and provides a broader range of services. A complementary approach aiming at the specification and investigation of the interrelations between calculi has been described in [21]. There, the calculi are defined in the algebraic specification language CASL. We believe that a toolbox like SparQ can provide a useful interface between the theoretical specification framework and the application areas of spatial cognition, like cognitive modeling or GIS.

In its current version, SparQ mainly focuses on calculi from the area of reasoning about the orientation of point objects or line segments. However, specifying and adding other calculi is simple. We hope to encourage researchers from other groups to incorporate their calculi in a community effort of providing a rich spatial reasoning environment. SparQ is designed as an open framework of single program components with text-based communication. It therefore allows for integrating code written in virtually any programming language, so that already existing code can easily be integrated into SparQ.

[1] **Spa**tial **R**easoning done **Q**ualitatively.

Specifically, the goals of SparQ are the following:

- providing reference implementations for spatial calculi from the QSR community
- making it easy to specify and integrate new calculi
- providing typical procedures required to apply QSR in a convenient way
- offering a uniform interface that supports switching between calculi
- being easily integrable into own applications

The current version of SparQ and further documentation will be made available at the SparQ homepage[2]. In the present text, we will describe SparQ and its utilization. The next section briefly recapitulates the relevant terms concerning QSR and spatial calculi as needed for the remainder of the text. In Section 3, we describe the services provided by SparQ. Section 4 explains how new calculi can be incorporated into SparQ, and Section 5 describes how SparQ can be integrated into applications. Finally, Section 6 contains a case study in which SparQ is employed to compare different calculi with respect to their ability of detecting the inconsistency in the Indian Tent Problem [22].

2 Reasoning with Qualitative Spatial Relations

A qualitative spatial calculus defines operations on a finite set $\mathcal{R}$ of spatial relations. The spatial relations are defined over a particular set of spatial objects, the domain D. In the rest of the text, we will encounter the sets of points in the plane, of oriented line segments in the plane, and of oriented points in the plane as domains. While a *binary calculus* deals with binary relations $R \subseteq D \times D$, a *ternary calculus* operates with ternary relations $R \subseteq D \times D \times D$.

The set of relations $\mathcal{R}$ of a spatial calculus is typically derived from a jointly exhaustive and pairwise disjoint (JEPD) set of *base relations* $\mathcal{BR}$ so that each pair of objects from D is contained in exactly one relation from $\mathcal{BR}$. Every relation in $\mathcal{R}$ is a union of base relations. Since spatial calculi are typically used for constraint reasoning and unions of relations correspond to disjunctions of relational constraints, it is common to speak of disjunctions of relations as well and write them as sets $\{B_1, ..., B_n\}$ of base relations. Using this convention, $\mathcal{R}$ is either taken to be the powerset $2^{\mathcal{BR}}$ of the base relations or a subset of the powerset. In order to be usable for constraint reasoning, $\mathcal{R}$ should contain at least the base relations B_i, the empty relation $\emptyset$, the universal relation U, and the identity relation Id. $\mathcal{R}$ should also be closed under the operations defined in the following.

As the relations are subsets of tuples from the same Cartesian product, the set operations union, intersection, and complement can be directly applied:

Union: $R \cup S = \{\ t \mid t \in R \vee t \in S\ \}$
Intersection: $R \cap S = \{\ t \mid t \in R \wedge t \in S\ \}$
Complement: $\overline{R} = U \setminus R = \{\ t \mid t \in U \wedge t \notin R\ \}$

[2] `http://www.sfbtr8.uni-bremen.de/project/r3/sparq/`

where R and S are both n-ary relations on D and t is an n-tuple of elements from D. The other operations depend on the arity of the calculus.

2.1 Operations for Binary Calculi

For binary calculi the other two important operations are conversion and composition:

Converse: $R^{\smile} = \{ (y, x) \mid (x, y) \in R \}$
(Strong) composition: $R \circ S = \{ (x, z) \mid \exists y \in D : ((x, y) \in R \wedge (y, z) \in S) \}$

For some calculi, no finite set of relations exists that includes the base relations and is closed under composition as defined above. In this case, a weak composition is defined instead that takes the union of all base relations that have a non-empty intersection with the result of the strong composition:

Weak composition: $R \circ_{weak} S = \{ B_i \mid B_i \in \mathcal{BR} \wedge B_i \cap (R \circ S) \neq \emptyset \}$

2.2 Operations for Ternary Calculi

While there is only one possibility to permute the two objects of a binary relation which corresponds to the converse operation, there exist 5 such permutations for the three objects of a ternary relation[3], namely [23]:

Inverse: $\mathrm{INV}(R) = \{ (y, x, z) \mid (x, y, z) \in R \}$
Short cut: $\mathrm{SC}(R) = \{ (x, z, y) \mid (x, y, z) \in R \}$
Inverse short cut: $\mathrm{SCI}(R) = \{ (z, x, y) \mid (x, y, z) \in R \}$
Homing: $\mathrm{HM}(R) = \{ (y, z, x) \mid (x, y, z) \in R \}$
Inverse homing: $\mathrm{HMI}(R) = \{ (z, y, x) \mid (x, y, z) \in R \}$

Composition for ternary calculi is defined according to the binary case:

(Strong) comp: $R \circ S = \{ (w, x, z) \mid \exists y \in D : ((w, x, y) \in R \wedge (x, y, z) \in S) \}$

Other ways of composing two ternary relations can be expressed as a combination of the unary permutation operations and the composition [24] and thus do not have to be defined separately. The definition of weak composition is identical to the binary case.

2.3 Constraint Reasoning with Spatial Calculi

Spatial calculi are often used to formulate constraints about the spatial configurations of a set of objects from the domain of the calculus as a constraint satisfaction problem (CSP): Such a spatial constraint satisfaction problem then consists of a set of variables $X_1, ..., X_n$ (one for each spatial object) and a set of constraints $C_1, ..., C_m$ which are relations from the calculus. Each variable X_i

[3] In general, two operations (permutation and rotation) are sufficient to generate all permuations (cmp. [20]). Therefore, not all of these operations need to be specified.

can take values from the domain of the utilized calculus. CSPs are often described as constraint networks which are complete labeled graphs with a node for each variable and each edge labeled with the corresponding relation from the calculus. A CSP is consistent, if an assignment for all variables to values of the domain can be found, that satisfies all the constraints. Spatial CSPs usually have infinite domains and thus backtracking over the domains can not be used to determine consistency.

Besides consistency, weaker forms of consistency called *local consistencies* are of interest in QSR. On the one hand, they can be employed as a forward checking technique reducing the CSP to a smaller equivalent CSP (one that has the same set of solutions). Furthermore, in some cases a form of local consistency can be proven to be not only necessary but also sufficient for consistency. If this is only the case for a certain subset $\mathcal{S} \subset \mathcal{R}$ and this subset exhaustively splits $\mathcal{R}$ (which means that every relation from $\mathcal{R}$ can be expressed as a disjunction of relations from $\mathcal{S}$), this at least allows to formulate a backtracking algorithm to determine consistency by recursively splitting the constraints and using the local consistency as a decision procedure for the resulting CSPs with constraints from $\mathcal{S}$ [25].

One important form of local consistency is *path-consistency* which (in binary CSPs) means that for every triple of variables each consistent evaluation of the first two variables can be extended to the third variable in such a way that all constraints are satisfied. Path-consistency can be enforced syntactically based on the composition operation (for instance with the algorithm by van Beek [26]) in $O(n^3)$ time where n is the number of variables. However, this syntactic procedure does not necessarily yield the correct result with respect to path-consistency as defined above. The same holds for syntactic procedures that compute other kinds of consistency. Whether syntactic consistency coincides with semantic consistency with respect to the domain needs be investigated for each calculus individually (see [27,28] for an in-depth discussion).

2.4 Supported Calculi

As mentioned above, qualitative calculi are based on a certain domain of basic entities: time intervals in the case of Allen's Interval Calculus [2], or objects like points, line segments, or regions in typical spatial calculi. In the following, we will briefly introduce those calculi that are currently included in SparQ and that will be used in the examples later on. A quick overview is given in Table 1 which also classifies the calculi according to their arity (binary, ternary), their domain (points, oriented points, line segments, regions), and the aspect of space modeled (orientation, distance, mereotopology).

FlipFlop Calculus (FFC) and the $\mathcal{LR}$ refinement. The FlipFlop calculus proposed in [9] describes the position of a point C (the referent) in the plane with respect to two other points A (the origin) and B (the relatum) as illustrated in Fig. 1. It can for instance be used to describe the spatial relation of C to B as seen from A. For configurations with $A \neq B$ the following base relations are distinguished: C can be to the **l**eft or to the **r**ight of the oriented line going through

Table 1. The calculi currently included in SparQ

Calculus	arity		domain				aspect of space		
	binary	ternary	point	or. point	line seg.	region	orient.	dist.	mereot.
FFC/$\mathcal{LR}$		√	√				√		
SCC		√	√				√		
DCC		√	√				√		
$\mathcal{DRA}_c$	√				√		√		
$\mathcal{OPRA}_m$	√			√			√		
RCC-5/8[4]	√					√			√

A and B, or C can be placed on the line resulting in one of the five relations **i**nside, **f**ront, **b**ack, **s**tart ($C = A$) or **e**nd ($C = B$) (cp. Fig. 1). Relations for the case where A and B coincide were not included in Ligozat's original definition [9]. This was done with the $\mathcal{LR}$ refinement [29] that introduces the relations **dou** ($A = B \neq C$) and **tri** ($A = B = C$) as additional relations, resulting in a total of 9 base relations. A $\mathcal{LR}$ relation $rel_{\mathcal{LR}}$ is written as $A, B\ rel_{\mathcal{LR}}\ C$, e.g. A, B **r** C as depicted in Fig. 1.

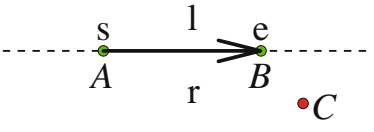
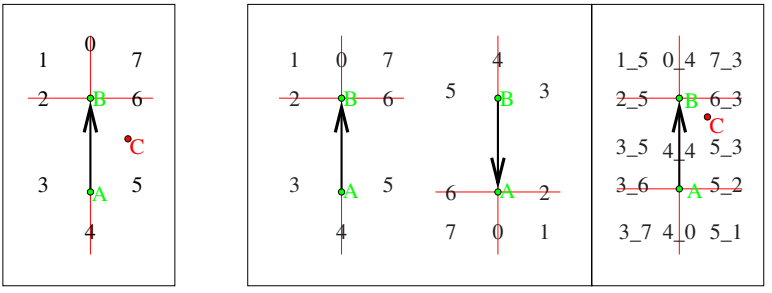


Fig. 1. The reference frame for the $\mathcal{LR}$ calculus, an refined version of the FlipFlop Calculus

Single Cross Calculus (SCC). The Single Cross Calculus is a ternary calculus that describes the direction of a point C (the referent) wrt. a point B (the relatum) as seen from a third point A (the origin). It was originally proposed in [6]. The plane is partitioned into regions by the line going through A and B and the perpendicular line through B. This results in eight distinct orientations as illustrated in Fig. 2(a). We denote these base relations by numbers from 0 to 7 instead of using linguistic prepositions, e.g. 2 instead of *left* as in [6]. Relations 0,2,4,6 are linear ones, while relations 1,3,5,7 are planar. In addition, three special relations exist for the cases $A \neq B = C$ (**bc**), $A = B \neq C$ (**dou**), and $A = B = C$ (**tri**). A Single Cross relation rel_{SCC} is written as $A, B\ rel_{SCC}\ C$, e.g. A, B **4** C or A, B **dou** C. The relation depicted in Fig. 2(a) is the relation A, B **5** C.

[4] Currently only the relational specification is available for RCC, but no 'qualify' module (cmp. Section 3.1).



(a) Single Cross Calculus reference frame

(b) The two Single Cross reference frames resulting in the overall Double Cross Calculus reference frame

Fig. 2. The Single and Double Cross reference systems

Double Cross Calculus (DCC). The Double Cross calculus [6] can be seen as an extension of the Single Cross calculus adding another perpendicular, this time at A (see Fig. 2(b) (right)). It can also be interpreted as the combination of two Single Cross relations, the first describing the position of C wrt. B as seen from A and the second wrt. A as seen from B (cf. Fig. 2(b) (left)). The resulting partition distinguishes 13 relations (7 linear and 6 planar) denoted by tuples derived from the two underlying SCC reference frames and four special cases, $A = C \neq B$ (**4_a**), $A \neq B = C$ (**b_4**), $A = B \neq C$ (**dou**), and $A = B = C$ (**tri**), resulting in 17 base relations overall. Fig. 2(b) depicts the relation A, B **5_3** C.

Coarse-grained Dipole Relation Algebra ($\mathcal{DRA}_c$). A dipole is an oriented line segment, e.g. as determined by a start and an end point. We will write $\boldsymbol{d}_{AB}$ for a dipole defined by start point A and end point B. The idea of using dipoles was first introduced by Schlieder [13] and extended in [14].

In the coarse-grained variant of the Dipole Calculus ($\mathcal{DRA}_c$) describes the orientation relation between two dipoles $\boldsymbol{d}_{AB}$ and $\boldsymbol{d}_{CD}$ with the preliminary that A, B, C, and D are in general position, i.e. no three disjoint points are collinear. Each base relation is a 4-tuple (r_1, r_2, r_3, r_4) of FlipFlop relations relating a point from one of the dipoles with the other dipole. r_1 describes the relation of C wrt. the dipole $\boldsymbol{d}_{AB}$, r_2 of D wrt. $\boldsymbol{d}_{AB}$, r_3 of A wrt. $\boldsymbol{d}_{CD}$, and r_4 of B wrt. $\boldsymbol{d}_{CD}$. The distinguished FlipFlop relations are **l**eft, **r**ight, **s**tart, and **e**nd (see Fig. 1). Dipole relations are usually written without commas and parentheses, e.g. *rrll*. Thus, the example in Fig. 3 shows the relation $\boldsymbol{d}_{AB}$ *rlll* $\boldsymbol{d}_{CD}$. Since the underlying points for a $\mathcal{DRA}_c$ relation need to be in general position the r_i can only take the values **l**eft, **r**ight, **s**tart, or **e**nd resulting in 24 base relations.

Oriented Point Relation Algebra $\mathcal{OPRA}_m$. The $\mathcal{OPRA}_m$ calculus [11] operates on oriented points. An oriented point is a point in the plane with an additional direction parameter. $\mathcal{OPRA}_m$ relates two oriented points $\boldsymbol{A}$ and $\boldsymbol{B}$ and describes their relative orientation towards each other. The granularity

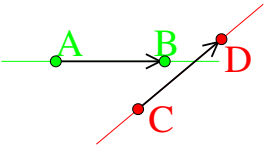
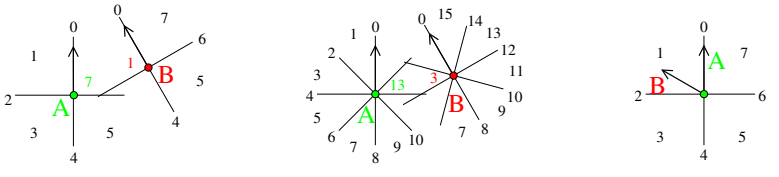


Fig. 3. A dipole configuration: $\boldsymbol{d}_{AB}$ *rlll* $\boldsymbol{d}_{CD}$ in the coarse-grained Dipole Relation Algebra ($\mathcal{DRA}_c$)

factor $m \in \mathbb{N}$ determines the number of distinguished relations. For each of the two oriented points, m lines are used to partition the plane into $2m$ planar and $2m$ linear regions. Fig. 4 shows the partitions for the cases $m = 2$ (Fig. 4(a)) and $m = 4$ (Fig. 4(b)). The orientation of the two points is depicted by the arrows starting at $\boldsymbol{A}$ and $\boldsymbol{B}$, respectively. The regions are numbered from 0 to $(4m-1)$. Region 0 always coincides with the orientation of the point. An $\mathcal{OPRA}_m$ relation $rel_{\mathcal{OPRA}_m}$ consist of pairs (i, j) where i is the number of the region of $\boldsymbol{A}$ which contains $\boldsymbol{B}$, while j is the number of the region of $\boldsymbol{B}$ that contains $\boldsymbol{A}$. These relations are usually written as $\boldsymbol{A}\ {}_m\angle_i^j\ \boldsymbol{B}$ with $i, j \in \mathcal{Z}_{4m}$[5]. Thus, the examples in Fig. 4 depict the relations $\boldsymbol{A}\ {}_2\angle_7^1\ \boldsymbol{B}$ and $\boldsymbol{A}\ {}_4\angle_{13}^3\ \boldsymbol{B}$. Additional relations describe situations in which both oriented points coincide. In these cases, the relation is determined by the number s of the region of $\boldsymbol{A}$ into which the orientation arrow of $\boldsymbol{B}$ falls (as illustrated in Fig. 4(c)). These relations are written as $\boldsymbol{A}\ {}_2\angle s\ \boldsymbol{B}$ ($\boldsymbol{A}\ {}_2\angle 1\ \boldsymbol{B}$ in the example).



(a) with granularity $m = 2$: $\boldsymbol{A}\ {}_2\angle_7^1\ \boldsymbol{B}$ (b) with granularity $m = 4$: $\boldsymbol{A}\ {}_4\angle_{13}^3\ \boldsymbol{B}$ (c) case where $\boldsymbol{A}$ and $\boldsymbol{B}$ coincide: $\boldsymbol{A}\ {}_2\angle 1\ \boldsymbol{B}$

Fig. 4. Two oriented points related at different granularities

3 SparQ

SparQ consists of a set of modules that provide different services required for QSR that will be explained below. These modules are glued together by a central script that can either be used directly from the console or included into own applications via TCP/IP streams in a server/client fashion (see Section 5). The general architecture is visualized in Fig. 5.

[5] $\mathcal{Z}_{4m}$ defines a cyclic group with $4m$ elements.

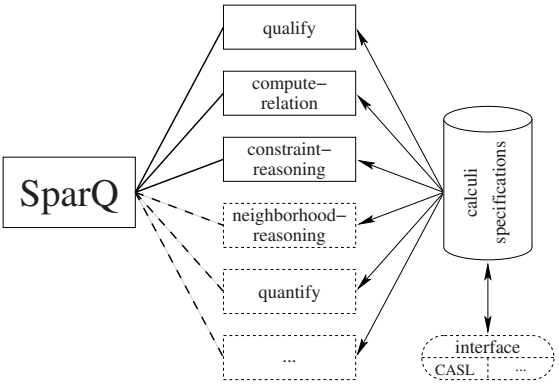


Fig. 5. Module architecture of the SparQ toolbox

The general syntax for using the SparQ main script is as follows:

```
$ ./sparq <module> <calculus identifier> <module-specific parameters>
```

Example:

```
$ ./sparq compute-relation dra-24 complement "(lrll llrr)"
```

where 'compute-relation' is the name of the module to be utilized, in this case the module for conducting operations on relations, 'dra-24' is the SparQ identifier for the dipole calculus $\mathcal{DRA}_c$, and the rest are module-specific parameters, here the name of the operation that should be conducted ('complement') and a string parameter representing the disjunction of the two dipole base relations *lrll* and *llrr*[6]. The example call thus computes the complement of the disjunction of these two relations.

Some calculi have calculus-specific parameters, for example the granularity parameter in $\mathcal{OPRA}_m$. These parameters are appended with a '`-`' after the calculus' base identifier. `opra-3` for example refers to $\mathcal{OPRA}_3$.

SparQ currently provides the following modules:

qualify transforms a quantitative geometric description of a spatial configuration into a qualitative description based on one of the supported spatial calculi

compute-relation applies the operations defined in the calculi specifications (intersection, union, complement, converse, composition, etc.) to a set of spatial relations

constraint-reasoning performs computations on constraint networks

Further modules are planned in future extensions. They comprise a quantification module for turning qualitative scene descriptions back into quantitative

[6] Disjunctions of base relations are always represented as a space-separated list of the base relations enclosed in parentheses in SparQ.

geometric descriptions and a module for neighborhood-based spatial reasoning. In the following section we will take a closer look at the three existing modules.

3.1 Scene Descriptions and Qualification

The purpose of the 'qualify' module is to turn a quantitative geometric scene description into a qualitative scene description with respect to a particular calculus. Calculi are specified via the calculus identifier that is passed with the call to SparQ. Qualification is required for applications in which one wants to perform qualitative computations over objects represented by their geometric parameters.

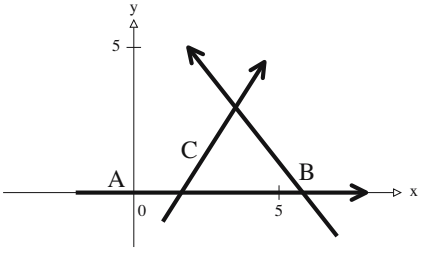


Fig. 6. An example configuration of three dipoles

The 'qualify' module reads a quantitative scene description and generates a qualitative one. A quantitative scene description is a list of base object descriptions (separated by spaces and enclosed in parentheses). Each base object description is a tuple consisting of an object identifier and object parameters that depend on the type of the object. For instance, let us say we are working with dipoles, i.e. oriented line segments. The object description of a dipole has the form '(name x_s y_s x_e y_e)', where name is the identifier of this particular dipole object and the rest are the coordinates of start and end point of the dipole. Let us consider the example in Fig. 6 which shows three dipoles A, B, and C. The quantitative scene description for this situation is:

```
( (A -2 0 8 0) (B 7 -2 2 5) (C 1 -1 4.5 4.5) )
```

The 'qualify' module has one module-specific parameter that needs to be specified:

mode: This parameter controls which relations are included into the qualitative scene description: If 'all' is passed as parameter, the relations between each pair of objects will be determined. If it is 'first2all' only the relations between the first and all other objects are computed.

The resulting qualitative scene description is a list of relation tuples (again separated by spaces and enclosed in parentheses). A relation tuple consists of the

object identifier of the relatum followed by a relation and the object identifier of the referent, meaning that the first object stands in this particular relation with the second object. The command to produce the qualitative scene description followed by the result is[7]:

```
$ ./sparq qualify dra-24 all
$ ( (A -2 0 8 0) (B 7 -2 2 5) (C 1 -1 4.5 4.5) )
> ( (A rllr B) (A rllr C) (B lrrl C) )
```

If we had chosen 'first2all' as mode parameter the relation between B and C would not have been included in the qualitative scene description.

3.2 Computing with Relations

The 'compute-relation' module realizes computations with the operations defined in the calculus specification. The module-specific parameters are the operation that should be conducted and one or more input relations depending on the arity of the operation. Assume we want to compute the converse of the dipole relation *llrl*. The corresponding call to SparQ and the result are:

```
$ ./sparq compute-relation dra-24 converse llrl
> (rlll)
```

The result is always a list of relations as operations often yield a disjunction of base relations. In the example above, the list contains a single relation. The composition of two relations requires one more relation as parameter because it is a binary operation, e.g.:

```
$ ./sparq compute-relation dra-24 composition llrr rllr
> (lrrr llrr rlrr slsr lllr rllr rlll ells llll lrll)
```

Here the result is a disjunction of 10 base relations. It is also possible to have disjunctions of base relations as input parameters. For instance, the following call computes the intersection of two disjunctions:

```
$ ./sparq compute-relation dra-24 intersection "(rrrr rrll rllr)"
  "(llll rrll)"
> (rrll)
```

3.3 Constraint Reasoning

The 'constraint-reasoning' module reads a description of a constraint network; this is a qualitative scene description that may include disjunctions and may be inconsistent and/or underspecified. It performs a particular kind of consistency check[8]. Which type of consistency check is executed depends on the first module specific parameter:

[7] In all the examples, input lines start with '$'. Output of SparQ is marked with '>'.

[8] The 'constraint-reasoning' module also provides some basic actions to manipulate constraint networks that are not further explained in this text. One example is the 'merge' operation that is used in the example in Section 5 (see the SparQ manual for details [30]).

action: The two consistency checks currently provided are 'path-consistency' and 'scenario-consistency'; the parameter determines which kind of consistency check is performed.

The action 'path-consistency' causes the module to enforce path-consistency on the constraint network using van Beek's algorithm [26] or to detect an inconsistency of the network in the process. In case of a ternary calculus the canonical extension of van Beek's algorithm described in [31] is used. For instance, we could check if the scene description generated by the 'qualify' module in Section 3.1 is path-consistent—which of course it is. To make the test slightly more interesting we add the base relation *ells* to the constraint between A and C; this results in a constraint network that is not path-consistent:

```
$ ./sparq constraint-reasoning dra-24 path-consistency
$ ( (A rllr B) (A (ells rllr) C) (B lrrl C) )
> Modified network.
> ( (B (lrrl) C) (A (rllr) C) (A (rllr) B) )
```

The result is a path-consistent constraint network in which *ells* has been removed. The output 'Modified network' indicates that the original network was not path-consistent and had to be changed. Otherwise, the result would have started with 'Unmodified network'. In the next example we remove the relation *rllr* from the disjunction. This results in a constraint network that cannot be made path-consistent; this implies that it is not consistent.

```
$ ./sparq constraint-reasoning dra-24 path-consistency
$ ( (A rllr B) (A ells C) (B lrrl C) )
> Not consistent.
> ( (B (lrrl) C) (A () C) (A (rllr) B) )
```

SparQ correctly determines that the network is inconsistent and returns the constraint network in the state in which the inconsistency showed up (indicated by the empty relation () between A and C).

In a last path-consistency example we use the ternary Double Cross Calculus:

```
$ ./sparq constraint-reasoning dcc path-consistency
$ ( (A B (7_3 6_3) C) (B C (7_3 6_3 5_3) D) (A B (3_6 3_7) D) )
> Not consistent.
> ( (A B () D) (A B (6_3 7_3) C) (B C (5_3 6_3 7_3) D) (D C (3_7) A) )
```

If 'scenario-consistency' is provided as argument, the 'constraint-reasoning' module checks if a path-consistent scenario exists for the given network. It uses a backtracking algorithm to generate all possible scenarios and checks them for path-consistency as described above. A second module-specific parameter determines what is returned as the result of the search:

return: This parameter determines what is to be returned in case of a constraint network for which path-consistent scenarios can be found. 'First' returns the first path-consistent scenario, 'all' returns all path-consistent scenarios, and 'interactive' returns one solution and allows to ask for the next solution until all solutions have been generated.

Path-consistency is also used as a forward-checking method during the search to make the search more efficient. For certain calculi, the existence of a path-consistent scenario implies consistency. However, this again has to be investigated for each calculus (cmp. Section 2.3). In the following example, we use 'first' as additional parameter so that only the first solution is returned:

```
$ ./sparq constraint-reasoning dra-24 scenario-consistency first
$ ( (A rele C) (A ells B) (C errs B) (D srsl C) (A rser D) (D rrrl B) )
> ( (B (rlrr) D) (C (slsr) D) (C (errs) B) (A (rser) D) (A (ells) B)
  (A (rele) C) )
```

In case of an inconsistent constraint network, SparQ returns 'Not consistent.'. As a future extension, we plan to allow specification of splitting subsets of a calculus for which path-consistency implies consistency. A splitting subset S will be used in a variant of the backtracking algorithm to decide consistency by searching for path-consistent instantiations that only contain relations from S.

4 Specifying Calculi in SparQ

For most calculi inclusion into SparQ should be straightforward. The main action to be taken is to provide the calculus specification. This is done in a Lisp-like syntax. Listing 1.1 shows an extract of the definition of a simple exemplary calculus for reasoning about distances between three point objects distinguishing the three relations 'closer', 'farther', and 'same'.

```
(def-calculus "Relative distance calculus (reldistcalculus)"
    :arity :ternary
    :base-relations (same closer farther)
    :identity-relation same

    :inverse-operation ((same same)
                        (closer closer)
                        (farther farther))

    :shortcut-operation ((same same)
                         (closer farther)
                         (farther closer))

    :composition-operation ((same same (same closer farther))
                            (same closer (same closer farther))
                            (same farther (same closer farther))
                            (closer same (same closer farther))
                            (closer closer (same closer farther))
                            [...]
```

Listing 1.1. Specification of a simple ternary calculus for reasoning about distances

The arity of the calculus, the base relations, the identity relation, and the different operations have to be specified, using lists enclosed in parentheses (e.g. when an operation returns a disjunction of base relations). In this example, the inverse operation applied to 'same' yields 'same', and composing 'closer' and 'same' results in the universal relation written as the disjunction of all base relations. As mentioned in Section 2.2, not all operations are required because some operations are combinations of other operations.

In addition to the calculus specification, it is necessary to provide the implementation of a qualifier function which for an n-ary calculus takes n geometric objects of the corresponding base type as input and returns the relation holding between these objects. The qualifier function encapsulates the methods for computing the qualitative relations from quantitative geometric descriptions. If it is not provided, the 'qualify' module will not work for this calculus.

For some calculi, it is not possible to provide operations in form of simple tables as in the example. For instance, $\mathcal{OPRA}_m$ has an additional parameter that specifies the granularity of the calculus and influences the number of base relations. Thus, the operations can only be provided in procedural form; this means the result of the operations are computed from the input relations when they are required. For these cases, SparQ allows providing the operations as implemented functions and uses a caching mechanism to store often required results.

5 Integrating SparQ into Own Applications

SparQ can also run in server mode which makes it easy to integrate it into applications. We have chosen a client/server approach as it allows for straightforward integration independently of the programming language used for implementing the application.

When run in server mode, SparQ takes TCP/IP connections and interacts with the client via simple plain-text line-based communication. This means the client sends commands which consist of everything following the './sparq' in the examples in this text and can then read the results from the TCP/IP stream.

```
# connect to sparq server on localhost, port 4443
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect(('localhost', 4443))
sockfile = sock.makefile('r')

# qualify a geometrical scenario with DRA-24
sock.send('qualify dra-24 first2all  ')
sock.send('((A 4 6 9 0.5) (B -5 5 0 2) (C -4 5 6 0))')
scene = readline()   # read the answer
print scene

# add an additional relation (B eses C)
sock.send("constraint-reasoning dra-24 merge")
sock.send(scene + '(B eses C)')
scene2 = readline()   # read the answer
print scene2

# check the new scenario for consistency
sock.send('constraint-reasoning dra-24 path-consistency')
sock.send(scene2)
print readline()   # print the answer
print readline()   # print the resulting constraint network
```

Listing 1.2. Integrating SparQ into own applications: an example in Python

SparQ is started in server mode by providing the command line option `--interactive` (`-i`), optionally followed by `--port` (`-p`) to specify the port.

```
$ ./sparq --interactive --port 4443
```

If no port is given, SparQ interacts with standard-input and standard-output, i.e., it can be used interactively from the shell.

An example is given in Listing 1.2 which shows a small Python program that opens a connection to the server and performs some simple computations (qualification, adding another relation, checking for path-consistency). It produces the following output:

```
> ( (A rrll B) (A rrll C) )
> ( (A rrll B) (A rrll C) (B eses C) )
> Not consistent.
> ( (B (eses) C) (A () C) (A (rrll) B) )
```

6 A Case Study: Using SparQ for the Indian Tent Problem

In this section we want to demonstrate the application of SparQ to a problem that can be seen as a kind of benchmark in QSR, the so-called *Indian Tent Problem*.

6.1 Indian Tent Definition

The Indian Tent Problem was first discussed by Röhrig [22]. It describes a very simple configuration of points in the plane that is not consistent and can be used to compare spatial reasoning formalisms with respect to their ability to detect this inconsistency.

The Indian Tent consists of a clockwise oriented triangle $\triangle ABC$ and an additional point D (see Fig. 7). The following facts are given: "C is right of AB", "D is left of AB" and "D is right of CB". From these facts follows geometrically that "D is right of CA". Thus, adding the (obviously wrong) fact "D is left of CA" results in an inconsistency.

In the following, we will show how to use SparQ to compare the properties of different calculi. We will model the Indian Tent Problem with three calculi (FFC, DCC and $\mathcal{DRA}_c$) and demonstrate how the constraint-based reasoning abilities of SparQ can be used to gain insights about how well local consistencies like path-consistency or scenario-consistency approximate consistency.

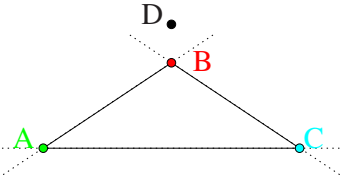
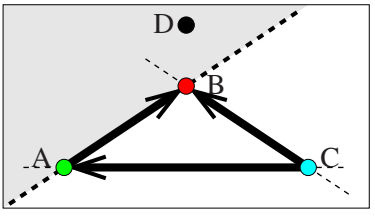
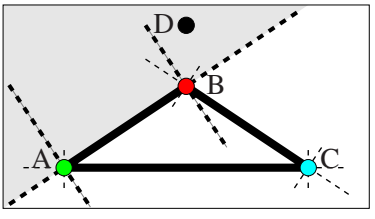


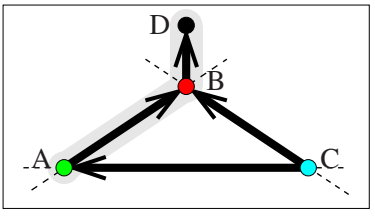
Fig. 7. The Indian Tent Problem



(A B l D)
(A B r C)
(C B r D)
(C A r D)



(A B (1_5 2_5 3_5 3_6 3_7) D)
(A B (5_1 5_2 5_3 6_3 7_3) C)
(C B (5_1 5_2 5_3 6_3 7_3) D)
(C A (5_1 5_2 5_3 6_3 7_3) D)



(AB rele CB)
(AB ells BD)
(CB errs BD)
(CA srsl CB)
(AB rser CA)
(CA rr?? BD)

Fig. 8. The Indian Tent problem modeled with the FlipFlop calculus, the Double Cross Calculus and $\mathcal{DRA}_c$. The relation printed in italics refers to the relation depicted by the gray region in the pictures. The '?' is a wildcard in SparQ which stands for an arbitrary valid symbol at this place, i.e., it denotes the disjunction of all possible relations.

6.2 Applying SparQ to the Indian Tent Problem

Fig. 8 shows consistent models of the Indian Tent Problem with FFC, DCC, and $\mathcal{DRA}_c$. For further use, we assume the models to be stored in simple text files with the labels 'FFC_consistent', 'DCC_consistent', and 'DRA_consistent'. We first want to check if we can enforce path-consistency on those models:

```
$ ./sparq constraint-reasoning ffc path-consistency < FFC_consistent
> Unmodified network.
> ( (A B (l) D) (A B (r) C) (C A (r) D) (C B (r) D) )

$ ./sparq constraint-reasoning dcc path-consistency < DCC_consistent
> Unmodified network.
> ( (A B (1_5 2_5 3_5 3_6 3_7) D) (A B (5_1 5_2 5_3 6_3 7_3) C)
    (C A (5_1 5_2 5_3 6_3 7_3) D) (C B (5_1 5_2 5_3 6_3 7_3) D) )
```

```
$ ./sparq constraint-reasoning dra-24 path-consistency < DRA_consistent
> Modified network.
> ( (BD (rlrr) CA) (CB (slsr) CA) (CB (errs) BD) (AB (rser) CA)
    (AB (ells) BD) (AB (rele) CB) )
```

Not surprisingly, SparQ can enforce path-consistency on all three models. Now we replace each last relation specified in Fig. 8 indicating that D is right of CA by the opposite relations '(C A (l) D)', '(C A (1_5 2_5 3_5 3_6 3_7) D)', and '(CA rl?? BD)', respectively. Thus, we create obviously inconsistent configurations which we assume to be stored in the files 'FFC_inconsistent', 'DCC_inconsistent', and 'DRA_inconsistent'. Now these are checked for path-consistency:

```
$ ./sparq constraint-reasoning ffc path-consistency < FFC_inconsistent
> Unmodified network.
> ( (A B (l) D) (A B (r) C) (C A (l) D) (C B (r) D) )

$ ./sparq constraint-reasoning dcc path-consistency < DCC_inconsistent
> Unmodified network.
> ( (A B (1_5 2_5 3_5 3_6 3_7) D) (A B (5_1 5_2 5_3 6_3 7_3) C)
    (C A (1_5 2_5 3_5 3_6 3_7) D) (C B (5_1 5_2 5_3 6_3 7_3) D) )

$ ./sparq constraint-reasoning dra-24 path-consistency < DRA_inconsistent
> Not consistent.
> ( (BD()CA) (CB (slsr) CA) (CB (errs) BD) (AB (rser) CA) (AB (ells) BD)
    (AB (rele) CB) )
```

The $\mathcal{DRA}_c$ model is found to be inconsistent. For FFC and DCC, however, SparQ shows that path-consistency can be enforced for the inconsistent models. For DCC this confirms the results of Röhrig [32]. So we also check the FFC and DCC models for scenario-consistency:

```
$ ./sparq constraint-reasoning ffc scenario-consistency first
  < FFC_inconsistent
> ( (A B (l) D) (A B (r) C) (C A (l) D) (C B (r) D) )

$ ./sparq constraint-reasoning dcc scenario-consistency first
  < DCC_inconsistent
> Not consistent.
```

This result shows that at least for this simple configuration scenario-consistency is sufficient to detect the inconsistency for DCC, while for FFC it is still not sufficient. As this example illustrates, SparQ can be a useful tool for experimentally comparing spatial calculi.

7 Conclusion and Outlook

The SparQ toolbox presented in this text is a first step towards making QSR techniques and spatial calculi accessible to a broader range of application developers. We hope that this initiative will catch interest in the QSR community

and will encourage researchers from other groups to incorporate their calculi into SparQ.

Besides including more calculi, extensions currently planned for SparQ are a module for neighborhood-based reasoning techniques [33,15] (e.g. for relaxing inconsistent constraint networks based on conceptual neighborhoods and for qualitative planning) and a module that allows quantification (turning a consistent qualitative scene description back into a geometric representation). This requires the mediation between the algebraic and geometric aspects of a spatial calculus together with the utilization of prototypes. Moreover, we want to include geometric reasoning techniques based on Gröbner bases as a service for calculus developers as these can be helpful, for instance, to derive composition tables [14]. The optimization of the algorithms included in SparQ is another issue that we want to pay more attention to in the future. Finally, we intend to incorporate interfaces that allow to exchange calculus specifications with other QSR frameworks (e.g. [21]).

Acknowledgements

The authors would like to thank three anonymous reviewers for valuable comments and suggestions. This research was carried out at the SFB/TR 8 Spatial Cognition supported by the German Research Foundation (DFG).

References

1. Cohn, A.G., Hazarika, S.M.: Qualitative spatial representation and reasoning: An overview. Fundamenta Informaticae 46(1-2), 1–29 (2001)
2. Allen, J.F.: Maintaining knowledge about temporal intervals. Communications of the ACM, 832–843 (1983)
3. Randell, D.A., Cui, Z., Cohn, A.: A spatial logic based on regions and connection. In: Nebel, B., Rich, C., Swartout, W. (eds.) Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference (KR 1992), pp. 165–176. Morgan Kaufmann, San Francisco (1992)
4. Egenhofer, M.J.: A formal definition of binary topological relationships. In: 3rd International Conference on Foundations of Data Organization and Algorithms, pp. 457–472. Springer, Heidelberg (1989)
5. Renz, J., Nebel, B.: On the complexity of qualitative spatial reasoning: A maximal tractable fragment of the region connection calculus. Artificial Intelligence 108(1-2), 69–123 (1999)
6. Freksa, C.: Using orientation information for qualitative spatial reasoning. In: Frank, A.U., Campari, I., Formentini, U. (eds.) Theories and methods of spatio-temporal reasoning in geographic space, pp. 162–178. Springer, Heidelberg (1992)
7. Frank, A.: Qualitative spatial reasoning about cardinal directions. In: Proceedings of the American Congress on Surveying and Mapping ACSM-ASPRS, Baltimore, Maryland, USA, pp. 148–167 (1991)
8. Ligozat, G.: Reasoning about cardinal directions. Journal of Visual Languages and Computing 9, 23–44 (1998)

9. Ligozat, G.: Qualitative triangulation for spatial reasoning. In: Campari, I., Frank, A.U. (eds.) COSIT 1993. LNCS, vol. 716, pp. 54–68. Springer, Heidelberg (1993)
10. Moratz, R., Nebel, B., Freksa, C.: Qualitative spatial reasoning about relative position: The tradeoff between strong formal properties and successful reasoning about route graphs. In: Freksa, C., Brauer, W., Habel, C., Wender, K.F. (eds.) Spatial Cognition III. LNCS (LNAI), vol. 2685, pp. 385–400. Springer, Heidelberg (2003)
11. Moratz, R., Dylla, F., Frommberger, L.: A relative orientation algebra with adjustable granularity. In: Proceedings of the Workshop on Agents in Real-Time and Dynamic Environments (IJCAI 2005) (2005)
12. Renz, J., Mitra, D.: Qualitative direction calculi with arbitrary granularity. [34]
13. Schlieder, C.: Reasoning about ordering. In: Kuhn, W., Frank, A.U. (eds.) COSIT 1995. LNCS, vol. 988, pp. 341–349. Springer, Heidelberg (1995)
14. Moratz, R., Renz, J., Wolter, D.: Qualitative spatial reasoning about line segments. In: Horn, W. (ed.) Proceedings of the 14th European Conference on Artificial Intelligence (ECAI), IOS Press, Berlin, Germany (2000)
15. Dylla, F., Moratz, R.: Exploiting qualitative spatial neighborhoods in the situation calculus. [35], pp. 304–322
16. Billen, R., Clementini, E.: A model for ternary projective relations between regions. In: Bertino, E., Christodoulakis, S., Plexousakis, D., Christophides, V., Koubarakis, M., Böhm, K., Ferrari, E. (eds.) EDBT 2004. LNCS, vol. 2992, pp. 310–328. Springer, Heidelberg (2004)
17. Goyal, R.K., Egenhofer, M.J.: Consistent queries over cardinal directions across different levels of detail. In: Tjoa, A.M., Wagner, R., Al-Zobaidie, A. (eds.) Proceedings of the 11th International Workshop on Database and Expert System Applications, Greenwich, pp. 867–880. IEEE Computer Society Press, Los Alamitos (2000)
18. Sharma, J.: Integrated Spatial Reasoning in Geographic Information Systems: Combining Topology and Direction. PhD thesis, University of Maine (1996)
19. Gerevini, A., Renz, J.: Combining topological and size information for spatial reasoning. Artificial Intelligence 137, 1–42 (2002)
20. Condotta, J.F., Ligozat, G., Saade, M.: A generic toolkit for n-ary qualitative temporal and spatial calculi. In: Proceedings of the 13th International Symposium on Temporal Representation and Reasoning (TIME 2006), Budapest, Hungary (2006)
21. Wölfl, S., Mossakowski, T.: CASL specifications of qualitative calculi. In: Cohn, A.G., Mark, D.M. (eds.) COSIT 2005. LNCS, vol. 3693, Springer, Heidelberg (2005)
22. Röhrig, R.: Representation and processing of qualitative orientation knowledge. In: Brewka, G., Habel, C., Nebel, B. (eds.) KI 1997: Advances in Artificial Intelligence. LNCS, vol. 1303, pp. 219–230. Springer, Heidelberg (1997)
23. Zimmermann, K., Freksa, C.: Qualitative spatial reasoning using orientation, distance, and path knowledge. Applied Intelligence 6, 49–58 (1996)
24. Scivos, A., Nebel, B.: Double-crossing: Decidability and computational complexity of a qualitative calculus for navigation. In: Montello, D.R. (ed.) COSIT 2001. LNCS, vol. 2205, Springer, Heidelberg (2001)
25. Ladkin, P., Reinefeld, A.: Effective solution of qualitative constraint problems. Artificial Intelligence 57, 105–124 (1992)
26. van Beek, P.: Reasoning about qualitative temporal information. Artificial Intelligence 58(1-3), 297–321 (1992)

27. Renz, J., Ligozat, G.: Weak composition for qualitative spatial and temporal reasoning. In: van Beek, P. (ed.) CP 2005. LNCS, vol. 3709, pp. 534–548. Springer, Heidelberg (2005)
28. Ligozat, G., Renz, J.: What is a qualitative calculus? A general framework [34]
29. Scivos, A., Nebel, B.: The finest of its class: The practical natural point-based ternary calculus $\mathcal{LR}$ for qualitative spatial reasoning [35], pp. 283–303
30. Wallgrün, J.O., Frommberger, L., Dylla, F., Wolter, D.: SparQ user manual v0.6. Technical Report 007-07/2006, SFB/TR 8 Spatial Cognition; Universität Bremen (2006)
31. Dylla, F., Moratz, R.: Empirical complexity issues of practical qualitative spatial reasoning about relative position. In: Workshop on Spatial and Temporal Reasoning at ECAI 2004, Valencia, Spain (2004)
32. Röhrig, R.: Repräsentation und Verarbeitung von qualitativem Orientierungswissen. PhD thesis, University of Hamburg (1998)
33. Freksa, C.: Temporal reasoning based on semi-intervals. Artificial Intelligence 1(54), 199–227 (1992)
34. Zhang, C., Guesgen, H.W., Yeap, W.-K. (eds.): PRICAI 2004. LNCS (LNAI), vol. 3157. Springer, Heidelberg (2004)
35. Freksa, C., Knauff, M., Krieg-Brückner, B., Nebel, B., Barkowsky, T. (eds.): Spatial Cognition IV. LNCS (LNAI), vol. 3343. Springer, Heidelberg (2005)