Developments in Spatio-Temporal Query Languages*

Martin Erwig & Markus Schneider
FernUniversität Hagen, Praktische Informatik IV
58084 Hagen, Germany
[erwig|markus.schneider]@fernuni-hagen.de

Abstract

Integrating spatio-temporal data as abstract data types into already existing data models is a promising approach to creating spatio-temporal query languages. In this context, an important new class of queries can be identified which is concerned with developments of spatial objects over time, that is, queries ask especially for changes in spatial relationships. Based on a definition of the notion of spatio-temporal predicate we provide a framework which allows to build more and more complex predicates starting with a small set of elementary ones. These predicates can be well used to characterize developments. We show how these concepts can be realized within the relational data model. In particular, we demonstrate how SQL can be extended to enable the querying of developments.

1 Introduction

Motivated by the deep relationships between temporal and spatial phenomena there has recently been an increased interest in designing *spatio-temporal data models* and *spatio-temporal databases*. A currently still open issue relates to the nature and definition of *spatio-temporal predicates* and their integration into *spatio-temporal query languages*. A query like "Was flight UA207 forced to cross a snow storm?" incorporates the feature of temporally changing spatial relationships and is, as we will show, very difficult and sometimes even impossible to express by using purely spatial predicates. It turns out that temporal query languages are also incapable of answering these queries.

In this paper we first give a definition of the notion of spatio-temporal predicate. Changes of spatial objects over time cause changes of their relationships. We show that *developments* of spatial relationships over time between spatial objects are a characteristic feature of spatio-temporal predicates. Second, we perform an integration of these predicates into SQL. We show that this integration profits very much from the abstract data type (ADT) approach for integrating complex objects into databases.

In Section 1.1 we briefly review currently available spatio-temporal data models and give reasons for our selection. In Section 1.2 we describe criteria for designing spatio-temporal query languages, and in Section 1.3 we relate spatio-temporal predicates to earlier work on spatial predicates.

1.1 Spatio-Temporal Data Models

So far, only a few data models for spatio-temporal data have been proposed. In [25] a spatial data model has been generalized to become spatio-temporal. Spatio-temporal objects are defined as so-called spatio-bitemporal complexes whose spatial features are described by simplicial complexes and whose temporal features are given by bitemporal elements attached to all components of simplicial complexes. On the other hand, temporal data models have been generalized to become spatio-temporal and include variants of Gadia's temporal model [16] which are described in [3, 2]. The main drawback of all these approaches is that they are incapable of modeling *continuous* changes of spatial objects over time.

Another approach which takes a more integrated view of space and time and which will be the basis of this paper introduces the concept of *spatio-temporal data types* [12, 13]. The definition of a temporal object in general is based on the observation that anything that changes over time can be expressed as a function over time. A temporal version of an object of type α is then given by a function from *time* to α. Spatio-temporal objects like *moving points* and *evolving regions* are regarded as special instances of temporal objects where α is a spatial data type.

Similar to the approach just mentioned, in [26, 27] based on the work in [20] *behavioral time sequences* are introduced. Each element of such a sequence contains a geometric value, a date, and a *behavioral function*, the latter

*This research was partially supported by the CHOROCHRONOS project, funded by the EU under the Training and Mobility of Researchers Programme, Contract No. ERB FMRX-CT96-0056.

describing the evolution up to the next element of the sequence. Time sequences could be used as representations for our temporal objects.

An issue that has been intensively discussed in temporal data modeling is whether a tuple timestamped or an attribute timestamped data model should be preferred. Tuple timestamped models (for example, [1, 6, 21]) expand the schema of a relation by one or more explicit temporal attributes that are used for describing the lifespan or validity period of a whole tuple. Each time an attribute of a tuple changes its value, the tuple has to be duplicated and modified. Hence, the information about an object is scattered over one or more relations. This approach impedes the modeling of continuous changes of spatial objects and the definition of corresponding operations and predicates; thus they are inappropriate for our purposes.

Instead of adding additional attributes to the relation schema, attribute timestamped models (for example, [5, 6, 16, 20]) aim at gathering information about an object into a single tuple and allow complex attribute values. These complex values incorporate the temporal dimension and are frequently modeled as functions from time into a value domain. From this perspective, attribute timestamped models are very similar to and fit very well with the models described in [13, 15].

However, the latter models additionally encapsulate (spatio-) temporal objects as ADT objects that can be integrated as complex values into databases [22, 23]. The ADT approach has several advantages. The first very important benefit is that employing ADTs is more expressive than relying on attribute timestamped models, let alone tuple timestamped models, since continuous changes can be modeled [13]. The second benefit is that a definition of ADT values is valid regardless of a particular DBMS data model and query language. The reason is that ADT values are *not* modeled by concepts of a DBMS data model and that they therefore do not depend on them. This facilitates an easy and elegant conceptual integration of ADT values into relational, complex object, object-oriented, or other data models and query languages.

1.2 Query Languages

Our intention is not to devise a new spatio-temporal query language from scratch but to appropriately extend the widespread database query language SQL. The main focus is on integrating developments.

We profit from the fact that the underlying data model rests on the ADT approach which necessitates only conservative extensions to SQL which are: (i) a set of basic spatio-temporal predicates, and (ii) an extension mechanism for new, more complex spatio-temporal predicates. We call this extended query language *STQL* (*Spatio-Temporal Query Language*). All added functionality is captured by ADT objects and operations. The benefit of this approach is the preservation of well known SQL concepts, the high-level treatment of spatio-temporal objects, and the easy incorporation of spatio-temporal predicates. Users can ask either standard SQL queries on standard data or use STQL features to inquire about situations involving spatio-temporal data.

The query facility of SQL is provided by the well known `SELECT-FROM-WHERE` clause. The integration of predicates like "<" or "<>" for standard data types such as integers or strings is well understood. In particular, there are only a few of them which allows one to include them as built-in predicates. When considering more complex and more structured data such as points, lines, or regions, one can try to systematically derive all reasonable predicates. The so-called 9-intersection model [9, 10] provides such canonical collections of predicates for each combination of spatial data types. For example, for two regions the eight predicates *disjoint*, *meet*, *overlap*, *coveredBy*, *covers*, *inside*, *contains*, and *equal* have been identified. A spatial query language based on these predicates and called *Spatial SQL* has been proposed in [7].

1.3 Spatio-Temporal Predicates

From an application point of view, we have found that expressing and querying temporal changes or *developments* of spatial objects is an important feature of a spatio-temporal query language. For this purpose *spatio-temporal predicates* are needed which model these developments and which can be used in the query part of STQL. A spatio-temporal predicate makes statements about the validity of the behavior of two spatio-temporal objects for some period of time.

In [14] we have given a definition of spatio-temporal predicates as a function taking two spatio-temporal objects as arguments and appropriately aggregating boolean values along the time axis by temporal quantifiers into a single boolean value. The problem is tackled on the basis of the 9-intersection model and on the basis of the work in [8] which considers *possible topological transitions* (that is, changes) of topological relationships.

Between a moving point and an evolving region we have identified a canonical collection of 28 spatio-temporal predicates, and between two evolving regions we have obtained not less than 2198 predicates [14]. The large numbers practically impede a naming of all these predicates and their reasonable employment from a user perspective. A first solution to this problem could be to furnish the user with a small, fixed, application-specific set of predicates. But possibly this is too restricted. An alternative could be to pursue a strategy like in [4]. There, an extension of the 9-intersection

model by additionally taking the dimension of the intersections into account leads to 52 possible relationships for all combinations of point, line, and region objects. The large number of predicates is reduced by grouping all topological cases into five overloaded topological predicates and by providing two boundary operators. These five predicates are mutually exclusive and capture all possible topological relationships. In our case, the number of predicates is much larger. Moreover, new predicates can be constructed from already existing ones. Hence, we advocate an extensible approach and provide a simple framework for composing spatio-temporal predicates. The integration into SQL becomes possible by an appropriate *macro mechanism*. This is similar to the way in which composite events are specified in [19]. The main difference is that events occur always at some instant in time whereas we also deal with predicates over whole time periods.

Section 2 presents a proposal for STQL. It describes the underlying data model and illustrates the embedding of spatio-temporal data types and operations into this language by query examples. Section 3 introduces spatio-temporal predicates and underpins the importance of developments for specifying changes of spatial relationships. Section 4 shows how developments are queried in STQL on the basis of spatio-temporal predicates. Section 5 sketches how queries containing spatio-temporal predicates could be evaluated. Section 6 draws some conclusions.

2 Extending SQL to STQL

In this section we introduce the first part of the design of a *spatio-temporal query language* called *STQL*. We first sketch the underlying data model before we show the embedding of spatio-temporal data types and operations into STQL by query examples.

2.1 The Data Model

For illustration purposes, we confine ourselves here to the well known relational model and to SQL as its most popular query language. A *relation scheme R* is written as $R(A_1 : D_1, \ldots, A_n : D_n)$ where the A_i are the *attributes* and the D_i are their respective value domains. For a relation $r : R(A_1 : D_1, \ldots, A_n : D_n)$ holds $r \subseteq D_1 \times D_2 \times \ldots \times D_n$. The domains can be standard types like integers, reals, booleans, or strings but also more complex types encapsulated into ADTs and including a comprehensive set of operations and predicates. Examples are spatial data types like points, lines, and regions [18] or graphs [11].

Similarly, we model spatio-temporal data as abstract data types which can be employed as attribute types in a relation. The relation itself has only a container function to store attribute data in tuples. The design of the model for spatio-temporal data is as follows: for compatibility with smoothly changing spatio-temporal objects we choose a continuous model of time, that is, $time = \mathbb{R}$. The temporal version of a value of type α that changes over time can be modeled as a *temporal function* of type $\tau(\alpha) = time \to \alpha$. We have used temporal functions as the basis of an algebraic data model for *spatio-temporal data types* [12] where α is assigned a spatial data type like *point* or *region*. For example, a point that changes its location over time is an element of type $\tau(point)$ and is called a *moving point*. Similarly, an element of type $\tau(region)$ is a region that can move and/or grow/shrink. It is called an *evolving region*. Currently, we do not consider a temporal version of lines, mainly because there seem to be not many applications of moving lines. A reason might be that lines are themselves abstractions or projections of movements and thus not the primary entities whose movements should be considered [13]. In any case, however, it is principally possible to integrate moving lines in much the same way as moving points if needed. In addition, we also have changing numbers and booleans, which are essential when defining operations on temporal objects. For instance, we could be interested in computing the (time-dependent) distance of an airplane and a storm. This could be achieved by an operation:

$$Distance : \tau(point) \times \tau(region) \to \tau(real)$$

In principle, we can take almost any non-temporal operation and "lift" it so that it works on temporal objects returning also a temporal object as a result. More precisely, for each function $f : \alpha_1 \times \ldots \times \alpha_n \to \beta$ its corresponding lifted version is defined by [17]:

$$\uparrow f : \tau(\alpha_1) \times \ldots \times \tau(\alpha_n) \to \tau(\beta)$$

with

$$\uparrow f(S_1, \ldots, S_n) := \{(t, f(S_1(t), \ldots, S_n(t))) \mid t \in time\}$$

For example, we have $Distance = \uparrow distance$. Note that this definition implies lifting also for constant objects of a non-temporal type α, that is

$$\uparrow : \alpha \to \tau(\alpha)$$

with

$$\uparrow c := \{(t, c) \mid t \in time\}$$

Temporal lifting is, of course, also applicable to spatial predicates. Consider the spatial predicate

$$inside : point \times region \to bool$$

The lifted version of this predicate has the type

$$\uparrow inside : Point \times Region \to Bool$$

with the meaning that it yields *true* for each time at which the point is inside the region, *undefined* whenever the point or the region is undefined, and *false* in all other cases.

To make notations more comprehensible we generally denote non-temporal types, entities, functions, and predicates by lower case letters while their temporal counterparts start with capital letters. For example, the spatial operation *distance* takes objects of type *point* and *region* and computes a number of type *real*, whereas its lifted version $Distance = \uparrow distance$ maps elements of type $Region = \tau(region)$ and $Point = \tau(point)$ to $Real = \tau(real)$ (temporal reals).

2.2 Querying with STQL

In the sequel, we introduce some operations and illustrate their embedding into STQL by posing some queries. As a scenario, we consider flights and weather conditions and use the two example relations

```
flights(id:string, Route:Point)
weather(kind:string, Extent:Region)
```

The attribute `id` identifies a flight, and `Route` records the route of a flight over time. The attribute `kind` classifies different weather events like hurricanes, high pressure areas, or snowfall, and `Extent` yields the evolving extent of each weather event.

Now we can ask queries like "Where was United Airlines flight 207 at time 8:00?".

```
SELECT Route(8:00) FROM flights
 WHERE id = "UA207"
```

This query shows the functional character of a spatio-temporal object by determining the value of the object at a certain time through a simple function application. A more general version of this query asks for the locations where the plane was between 7:00 am and 9:00 am.

```
SELECT trajectory(Route(7:00..9:00))
  FROM flights
 WHERE id = "UA207"
```

The ". ." notation specifies a range of time values, that is, a time interval. If a spatio-temporal object is applied to a time interval (or a collection of disjoint time intervals separated by commas), this expression yields a spatio-temporal object restricted to that time interval (function restriction). The `trajectory` operation computes the spatial projection of a spatio-temporal object onto the Euclidean plane. For a moving point it yields an object of type *line*. Note that isolated stationary points that can, in general, also occur are ignored. For an evolving region the trajectory operation returns a 2D region which results from projecting the union of the region values at all times onto the Euclidean plane.

The next query asks for values of the domain of a spatio-temporal object, that is, its lifespan: "When was a plane over the Eiffel Tower?"

```
SELECT dom(Intersection(
           Route, ^EiffelTower))
  FROM flights
```

The `Intersection` operator is lifted and computes the time-dependent intersection of two moving points. The result is again a moving point comprising all those (*time*, *point*)-pairs where the two original moving points met. We assume that `EiffelTower` describes a point containing the coordinates of the Eiffel Tower. In STQL the lifting operator is denoted by `^`. It is here applied to a point as a non-temporal object and yields a stationary point over time, that is, actually this moving point does not move. Afterwards, the `dom` operator collects the times when this intersection is defined. In this way inverse temporal functions can be computed.

The following query inquires about the largest snow areas at all times.

```
SELECT Area(max(Extent))
  FROM weather
 WHERE kind = "snow"
```

The query demonstrates an example of a *spatio-temporal aggregation* operation `max` which is an extension of the well known aggregation operator in SQL of the same name. It is here applied to a collection of evolving regions contained in a relation column and computes a new evolving region. Internally, this operator is based on a binary function max_{st} applied to two evolving regions R_1 and R_2 and yielding a new evolving region in the following way:

$$max_{st}(R_1, R_2) := \{(t, r) \mid t \in time \wedge r = max_{geo}(R_1(t), R_2(t))\}$$

This definition uses a function max_{geo} which is applied to two regions r_1 and r_2 and which returns the larger of both regions:

$$max_{geo}(r_1, r_2) = \begin{cases} r_1 & \text{if } area(r_1) > area(r_2) \\ r_2 & \text{otherwise} \end{cases}$$

Altogether this means that for n evolving regions $R_1, \ldots, R_n$ we first compute the evolving region $R = max_{st}(R_1, \ldots, max_{st}(R_{n-1}, R_n)\ldots)$. Afterwards, we apply the lifted operator `^area` to R which computes the area of R at all times. As the final result we obtain a temporal real.

Alternatively, we can answer the query by

```
SELECT max(Area(Extent))
  FROM weather
 WHERE kind = "snow"
```

Here, first the `^area` operator is applied to each snow area and returns a temporal real. Then the `max` operator takes the collection of temporal reals and produces a new temporal real by selecting the largest of all *real* values occurring in the temporal reals at each time.

In the Introduction we have emphasized that tuple timestamped models are too inflexible for our purposes and that we prefer attribute timestamped approaches. But even the latter approaches are restricted in the sense that they are incapable of modeling temporal developments of continuously evolving spatial objects. For example, consider the query "Determine the time when flight UA207 flew into a hurricane". Using a temporal query language like *TempSQL* [16] using attribute timestamps, we are not able to obtain a result, since first it cannot express and query any continuous developments but only stepwise changes and second there is no interpolation mechanism telling when an event happened within a time interval. But by employing our ADT approach, we can formulate the query as follows:

```
SELECT min(dom(
           Intersection(Route, Extent)))
  FROM flights, weather
 WHERE id = "UA207" AND
       kind = "hurricane"
```

The lifted `Intersection` operation is here applied to a moving point and an evolving region. It yields that part of `Route` lying inside `Extent`. The `min` operator here computes the minimum of all time values when the intersection is defined.

If we formulate this query a little bit more generally like "Determine the times when airplanes flew into hurricanes", a lot of plane-hurricane combinations might produce undesired null values because the flight and the hurricane just considered did not intersect at all. (Note that this is also possible in the more restricted first query.) In this case, the use of a spatio-temporal predicate in the `WHERE` clause could be very helpful and avoid these null values in the result relation. Such a predicate could investigate in advance whether a flight and a hurricane came into contact or not so that combinations not fulfilling this predicate are not taken into account any longer; their intersection is not computed. In the next section we will see how these predicates can be defined.

A second query showing the necessity of spatio-temporal predicates, in particular, for temporal developments, asks: "Determine the flights entering a hurricane". The problem here is that for each plane/hurricane combination we have to check the validity of different spatial predicates during a series of events and periods in a given temporal order. This means, we have to examine whether there has been a constellation when the plane and the hurricane were disjoint for a while, when afterwards they met at one point in time, and when finally the plane was inside the hurricane for a while. The development of entering a hurricane is only true if each of the three subqueries can be answered in the affirmative and if they have been temporally occurred one after the other. The series is like a specification that has to be matched at least once by each plane/hurricane combination.

```
SELECT id
  FROM flights, weather
 WHERE kind = "hurricane" AND
       not(Route(min(dom(Route))) inside
           Extent(min(dom(Route)))) AND
       Route(max(dom(Route))) inside
       Extent(max(dom(Route)))
```

Obviously, this query is very complicated. It works as follows: after the computation of the departure time of the flight (`min(dom(Route))`), the `Route` object is applied to this value and yields a point. The `Extent` object is applied to the same time value and yields a region. Using the spatial predicate `inside`, we check whether the point lies inside the region. If this is not true, we know that at the departure time of the flight the plane was outside of the hurricane. Similarly, we compute the arrival time of the flight (`max(dom(Route))`) and apply both `Route` and `Extent` to this time value. Again, we check whether the point lies inside the region, and if this is true, we know that `Route` must have entered the `Extent` object. This, in particular, implies that they met at the border of the hurricane. A limitation of this query is that we cannot determine whether plane and hurricane met only for one moment (straight entering) or whether the plane ran along the border for a while and then entered the hurricane (delayed entering). We will see how to express queries like these much more concisely in Section 4.

3 Spatio-Temporal Predicates and Developments

We have seen that with the ADT approach spatial, temporal, and even some spatio-temporal conditions can be expressed completely within "classical" SQL; no change to the syntax is needed. In this section we investigate the structure of spatio-temporal predicates in more detail, in particular, we will consider how to construct compound spatio-temporal predicates from basic ones. This will eventually lead to a modular and transparent mechanism for extending STQL by providing new spatio-temporal predicates through a small macro facility.

3.1 Spatio-Temporal Predicates

A spatio-temporal predicate is essentially a function that aggregates the values of a spatial predicate as it evolves

over time. In other words, a spatio-temporal predicate can be thought of as a lifted spatial predicate yielding a temporal boolean, which is aggregated by determining whether that temporal boolean was sometimes or always true. Thus, a spatio-temporal predicate is a function of type $\tau(\alpha) \times \tau(\beta) \to \mathbb{B}$ for $\alpha, \beta \in \{point, region\}$.

Consider again the definition of $\uparrow inside$ from Section 2.1. We can define two spatio-temporal predicates *sometimes-inside* and *always-inside* that yield true if $\uparrow inside$ yields true at some time, respectively, at all times.

Whereas the definition for *sometimes-inside* is certainly reasonable, the definition for *always-inside* is questionable, since it yields false whenever the point or the region is undefined. This is not what we would expect. For example, when the moving point has a shorter lifetime than the evolving region but is always inside the region, we would expect *always-inside* to yield true. Actually, we can distinguish different kinds of "forall" quantifications that result from different time intervals over which aggregation can be defined to range. In the case of *inside* the expected behavior is obtained if the aggregation ranges over the lifetime of the first argument, the moving point. This is not true for all spatial predicates. Actually, it depends on the nature and use of each individual predicate. For example, two spatio-temporal objects are considered as being *equal* only if they are equal on both objects' lifetimes, that is, the objects must have the same lifespans and must be always equal during these.

In order to be able to concisely build spatio-temporal predicates, we use the following general syntax: $Q_{op}.F$ where Q is either $\forall$ or $\exists$, *op* is a function mapping two sets into a new set,[1] and F is a formula built by logical connectives ($\vee$, $\wedge$, ...) and spatial and spatio-temporal predicates as constants. Such an expression then denotes the spatio-temporal predicate:

$$\lambda(S_1, S_2).Q\, t \in op(dom(S_1), dom(S_2)).[\![F]\!]$$

where $[\![F]\!]$ denotes the value of the formula, which is obtained by interpreting logical connectives as usual and computing for a spatial predicate p the value $p(S_1(t), S_2(t))$ and for a spatio-temporal predicate P the value $P(S_1, S_2)$. This means that, for example, $\forall_{\pi_1}.inside$ denotes the spatio-temporal predicate

$$\lambda(S_1, S_2).\forall\, t \in dom(S_1).inside(S_1(t), S_2(t))$$

In general, $\lambda(x_1, x_2, \ldots).e$ denotes a function that takes arguments $x_1, x_2, \ldots$ and returns a value determined by the expression e. So the above expression denotes a function that takes two arguments S_1 and S_2 and yields the boolean value denoted by the $\forall$-expression.

[1] For example, $\cap$ or π_1 that yield the intersection or the first set, respectively. If *op* is omitted, we define t to range over *time*.

With this notation we can give the definitions for the spatio-temporal versions of the eight basic spatial predicates (for two regions).

Definition 1

$$\begin{array}{ll} Disjoint & := \forall_{\cap}.disjoint \\ Meet & := \forall_{\cap}.meet \\ Overlap & := \forall_{\cap}.overlap \\ Equal & := \forall_{\cup}.equal \\ Covers & := \forall_{\pi_2}.covers \\ Contains & := \forall_{\pi_2}.contains \\ CoveredBy & := \forall_{\pi_1}.coveredBy \\ Inside & := \forall_{\pi_1}.inside \end{array}$$

For a moving point and a moving region we have just the three basic predicates *Disjoint*, *Meet*, and *Inside*, which are defined as above.

The chosen aggregations are motivated and discussed in detail in [14].

3.2 Developments: Specifications of Spatial Changes

In order to define compound spatio-temporal predicates, which capture the change of spatial situations, we need a way of restricting the temporal scope of basic spatio-temporal predicates. This becomes possible by predicate refinements (note that $S|_I$ denotes the partial function that yields $S(t)$ for all $t \in I$ and is undefined otherwise):

Definition 2 *Let I be a (half-) open or closed interval. Then* $P_I := \lambda(S_1, S_2).P(S_1|_I, S_2|_I)$.

When we now consider how spatial situations can change over time, we observe that certain relationships can be observed only for a period of time and not for only a single time point (given that the participating objects do exist for a period of time) while other relationships can hold at instants as well as on time intervals. Predicates that can hold at time points and intervals are: *equal*, *meet*, *covers*, *coveredBy*; these are called *instant predicates*. Predicates that can only hold on intervals are: *disjoint*, *overlap*, *inside*, *contains*; these are called *period predicates*.

It is interesting to note that (in satisfiable developments) instant and period predicates always occur in alternating patterns, for example, there cannot be two spatio-temporal objects that satisfy *Inside* immediately followed by *Disjoint*. In contrast, *Inside* first followed by *Meet* and then followed by *Disjoint* can be satisfied.

Now we can define three operations for elementary combinations of spatio-temporal and spatial predicates: $p \vdash P$ (*from*) defines a spatio-temporal predicate that at some time t_0 checks p and then enforces P for all $t > t_0$; $P \dashv p$ (*until*) is defined dually, that is, P must hold until p is true at

some time t_0. Finally, $P \dashv p \vdash Q$ (*then*) is true if there is some time t_0 when p is true so that P holds before t_0 and Q holds after t_0. In the following definition we abbreviate open intervals $]t,\infty[$ and $]\infty,t[$ by simply writing $>t$ and $<t$, respectively. (Note that variable t ranges over *time*.)

Definition 3 (Temporal Composition)

$$\begin{aligned} p \vdash P &:= \exists.p \wedge P|_{>t} \\ P \dashv p &:= \exists.p \wedge P|_{<t} \\ P \dashv p \vdash Q &:= \exists.p \wedge P|_{<t}(S_1,S_2) \wedge Q|_{>t}(S_1,S_2) \end{aligned}$$

It is easy to see that *then* is a combination of *from* and *until* and that temporal composition is associative:

$$\begin{aligned} P \dashv p \vdash Q &\iff P \dashv p \ \wedge \ p \vdash Q \\ P \dashv p \vdash (Q \dashv q \vdash R) &\iff (P \dashv p \vdash Q) \dashv q \vdash R \end{aligned}$$

In order to be able to denote developments concisely we use some syntactic sugar in writing down cascades of compositions. Recall that p (P) ranges over arbitrary spatial (spatio-temporal) predicates. In the table below we let I range over instant predicates, and i is assumed to be the corresponding spatial predicate from which the spatio-temporal I was derived (for example, if $I = Meet$, then $i = meet$).

Abbreviation	Expands to
$p \triangleright P$	$p \vdash P$
$P \triangleright p$	$P \dashv p$
$P \triangleright p \triangleright Q$	$P \dashv p \vdash Q$
$I \triangleright P$	$I \dashv i \vdash P$
$P \triangleright I$	$P \dashv i \vdash I$

Now we can define developments of spatio-temporal objects by simply sequencing basic spatio-temporal predicates. For example, we can define predicates for capturing the scenario of a point entering or crossing a region:

$$\begin{aligned} Enters &:= Disjoint \triangleright meet \triangleright Inside \\ Crosses &:= Disjoint \triangleright meet \triangleright Inside \triangleright meet \triangleright Disjoint \end{aligned}$$

However, sequential temporal composition is just one possibility to build new spatio-temporal predicates. Another important concept is to denote alternative developments:

Definition 4 (Temporal Alternative)

$$P|Q := \lambda(S_1,S_2).P(S_1,S_2) \vee Q(S_1,S_2)$$

(Note that we could also have used the shorter specification: $P|Q := \forall.P \vee Q$. However, this might be irritating, since the quantification is not relevant because only spatio-temporal predicates occur in the formula.) As an example, consider a moving point on a border of a region. The situations that can arise when the point leaves the border are captured by the alternative: *Disjoint*|*Inside*.

When we are interested only in an initial or a final part of a development, it is helpful to have a kind of "wildcard" spatio-temporal predicate that can be used to express "don't care" parts of developments. Therefore, we define the following predicate:

$$True := \lambda(S_1,S_2).true$$

(We require *True* to be an instant predicate; the corresponding spatial predicate is $\lambda(s_1,s_2).true$.) Note that *True* is a unit element with respect to $\triangleright$ and a zero element for $|$:

$$\begin{aligned} True \triangleright P &= P = P \triangleright True \\ True|P &= True = P|True \end{aligned}$$

An example for the use of *True* follows in the next section.

Finally, we define the "backward" (or "reverse") combinator ($P^{\leftarrow}$):

Definition 5

$$P^{\leftarrow} := \begin{cases} R^{\leftarrow} \triangleright Q^{\leftarrow} & \text{if } P = Q \triangleright R \\ Q^{\leftarrow}|R^{\leftarrow} & \text{if } P = Q|R \\ P & \text{otherwise} \end{cases}$$

From the definition it can be seen that the backward combinator enjoys the following law:

$$(P^{\leftarrow})^{\leftarrow} = P$$

4 Querying Developments in STQL

We extend SQL by (i) the set of eight basic spatio-temporal predicates and (ii) by a facility to compose new predicates.

Let us first reconsider the example query of finding out all planes that ran into a hurricane. With a predicate combinator `>>` that has the semantics of temporal composition $\triangleright$ we can formulate the query as:

```
SELECT id FROM flights, weather
 WHERE kind = "hurricane" AND
       Route Disjoint>>meet>>Inside Extent
```

Since some compound predicates we will be needed more frequently and since some of them have quite longish specifications, we introduce a kind of macro definition facility to introduce new predicates. The syntax is given below. As basic predicate (*p-basic*) we allow all the elementary spatio-temporal predicates defined in Section 3.1. The notation of predicate operators is taken as much as possible from the combinators defined in the preceding section.

```
p-def  →  DEFINE p-name AS p-expr
p-expr →  p-basic
       |  p-name
       |  p-expr >> p-expr
       |  p-expr | p-expr
       |  rev(p-expr)
```

We use the convention that | binds stronger than >> and that combinators >> and | bind stronger than predicate application. (Therefore, we could, for example, omit the brackets around the spatio-temporal predicate in the above example query.)

Now we can define a predicate `Enters` as follows:

```
DEFINE Enters AS Disjoint>>meet>>Inside
```

Hence, we can formulate the query asking for planes entering a hurricane simply as:

```
SELECT * FROM flights, weather
 WHERE kind = "hurricane" AND
       Route Enters Extent
```

As further examples consider the definition of the predicates `Crosses` and `Bypass`:

```
DEFINE Leaves AS rev(Enters)
DEFINE Crosses AS Enters>>Leaves
DEFINE Bypass AS Disjoint>>Meet>>Disjoint
```

Note that the predicate `Crosses` is equal to the definition given in Section 3.2 because `rev(Enters) = Inside>>meet>>Disjoint` and `Inside>>Inside = Inside`. General laws expressing relationships like these are given in [14].

We can use development predicates also within `GROUP BY` clauses, for example, we might be interested in the number of planes that were, respectively, were not entering snow storms or fog areas:

```
  SELECT COUNT(*) FROM flights, weather
   WHERE kind = "snow storm" OR
         kind = "fog"
GROUP BY Route Enters Extent
```

To demonstrate the use of developments on two evolving regions we assume we have two relations storing informations about forests and fires:

```
forest (name:string, Area:Region)
fire (fname:string, Extent:Region)
```

Now we could be interested, for example, in all forests that were completely destroyed by a particular fire. The fact that a forest is destroyed means that it is, at least from some time on, completely inside of (or equal to) the fire region, that is, after the fire is over, the forest does not exist anymore. But before that many different relationships between the fire and the forest are possible, for example, the fire ignition can happen within the forest, at its border or outside. Since we do not want to care about all these possibilities, we can use the predicate *True* as a wildcard preceding the final condition, which we denote in STQL by `_`.

```
SELECT name FROM forest, fire
 WHERE Area _>>Inside|Equal Extent
```

This means that for a certain period of time we do not care at all about the relationship between the forest and the fire, which is expressed by `_` that constantly yields *true*; we only require the existence of a time point after which `Inside` or `Equal` holds.

Finally, as an example for querying spatio-temporal developments of two moving points, suppose we have a relation recording migration of birds.

```
birds (swarm:string, Movement:Point)
```

We might be interested in swarms that fly together, then take different routes for some time and finally meet again. This can be expressed as an STQL query:

```
DEFINE Remeets
    AS _>>Meet>>Disjoint>>Meet>>_

SELECT A.swarm, B.swarm
  FROM birds (A), birds (B)
 WHERE A.Movement Remeets B.Movement
```

5 Query Evaluation

We shortly sketch how the evaluation of queries using complex spatio-temporal predicates can work. First, predicates are transformed into *development normal form*, that is, alternatives occurring as subparts nested within developments are moved to the outer level. This is possible, since $\triangleright$ distributes over $|$, that is, we know:

$$P \triangleright Q|R = (P \triangleright Q)|(P \triangleright R)$$
$$P|Q \triangleright R = (P \triangleright R)|(Q \triangleright R)$$

Thus, we can restrict ourselves to developments that do not contain alternatives.

Next we assume that evolving regions are represented as 3D volumes, and moving points are represented as 3D lines where we assume the z-axis represents *time* (for details, see [15]).

Now a development can be checked as follows. Consider, for example, the case for an evolving region and a moving point: we first compute the geometric intersection of both representations so that we obtain as a result a sequence of z-intervals together with the information about the corresponding parts of the volume/line (each of which might be undefined), of their intersection, and of their topological relationship, which can be one of "line inside volume", "line runs along volume border", or "line outside volume". The crucial point is that for each change in the topological relationship, a new interval is reported. For topological relationships that hold only for one single z-coordinate

(here, intersection points) no (degenerated) interval has to be reported, since it can be derived from the other intervals. The intersection procedure for two volumes is analogous.

Finally, what is left is to match the computed sequence against the development specification (which actually amounts to substring matching).

We are currently investigating several 3D intersection algorithms which we will describe in a subsequent paper.

6 Conclusions

Based on an ADT approach to the integration of spatio-temporal data types into data models we have shown how queries on spatio-temporal objects can be formulated within standard SQL. Observing that querying developments of spatial objects is of particular interest we have demonstrated how to define basic and compound spatio-temporal predicates as specifications for developments. Using complex developments within queries has been enabled by a modest extension to the DDL part of SQL.

References

[1] G. Ariav. An Overview of TQuel. *ACM Trans. on Database Systems*, 11(4):499–527, 1986.

[2] M. H. Böhlen, C. S. Jensen, and B. Skjellaug. Spatio-Temporal Database Support for Legacy Applications. In *ACM Symp. on Applied Computing*, pages 226–234, 1998.

[3] T. S. Cheng and S. K. Gadia. A Pattern Matching Language for Spatio-Temporal Databases. In *ACM Conf. on Information and Knowledge Management*, pages 288–295, 1994.

[4] E. Clementini, P. Felice, and P. Oosterom. A Small Set of Formal Topological Relationships Suitable for End-User Interaction. In *3rd Int. Symp. on Advances in Spatial Databases*, LNCS 692, pages 277–295, 1993.

[5] J. Clifford and A. Croker. The Historical Relational Data Model (HRDM) Revisited. *In [24]*, pages 6–27, 1993.

[6] J. Clifford, A. Croker, and A. Tuzhilin. On the Completeness of Query Languages for Grouped and Ungrouped Historical Data Models. *In [24]*, pages 496–533, 1993.

[7] M. J. Egenhofer. Spatial SQL: A Query and Presentation Language. *IEEE Transactions on Knowledge and Data Engineering*, 6(1):86–95, 1994.

[8] M. J. Egenhofer and K. K. Al-Taha. Reasoning about Gradual Changes of Topological Relationships. In *Int. Conf. GIS – From Space to Territory: Theories and Methods of Spatio-Temporal Reasoning in Geographic Space*, LNCS 639, pages 196–219, 1992.

[9] M. J. Egenhofer and R. D. Franzosa. Point-Set Topological Spatial Relations. *Int. Journal of Geographical Information Systems*, 5(2):161–174, 1991.

[10] M. J. Egenhofer and J. Herring. A Mathematical Framework for the Definition of Topological Relationships. In *4th Int. Symp. on Spatial Data Handling*, pages 803–813, 1990.

[11] M. Erwig and R. H. Güting. Explicit Graphs in a Functional Model for Spatial Databases. *IEEE Transactions on Knowledge and Data Engineering*, 5(6):787–804, 1994.

[12] M. Erwig, R. H. Güting, M. Schneider, and M. Vazirgiannis. Abstract and Discrete Modeling of Spatio-Temporal Data Types. In *6th ACM Symp. on Geographic Information Systems*, pages 131–136, 1998.

[13] M. Erwig, R. H. Güting, M. Schneider, and M. Vazirgiannis. Spatio-Temporal Data Types: An Approach to Modeling and Querying Moving Objects in Databases. *GeoInformatica*, 3(3), 1999. To appear.

[14] M. Erwig and M. Schneider. Spatio-Temporal Predicates. Technical Report, FernUniversität Hagen, 1999.

[15] M. Erwig, M. Schneider, and R. H. Güting. Temporal Objects for Spatio-Temporal Data Models and a Comparison of Their Representations. In *Int. Workshop on Advances in Database Technologies*, LNCS 1552, pages 454–465, 1998.

[16] S. K. Gadia and S. S. Nair. Temporal Databases: A Prelude to Parametric Data. *In [24]*, pages 28–66, 1993.

[17] R. H. Güting, M. H. Böhlen, M. Erwig, C. S. Jenssen, N. A. Lorentzos, M. Schneider, and M. Vazirgiannis. A Foundation for Representing and Querying Moving Objects. Technical Report 238, FernUniversität Hagen, 1998.

[18] R. H. Güting and M. Schneider. Realm-Based Spatial Data Types: The ROSE Algebra. *VLDB Journal*, 4(2):100–143, 1995.

[19] I. Motakis and C. Zaniolo. Composite Temporal Events in Active Databases: A Formal Semantics. In J. Clifford and A. Tuzhilin, editors, *Recent Advances in Temporal Databases*, pages 332–351. Springer Verlag, 1995.

[20] A. Segev and A. Shoshani. Logical Modeling of Temporal Data. *In [24]*, pages 248–270, 1993.

[21] R. Snodgrass. A Temporally Oriented Data Model. *In [24]*, pages 141–182, 1993.

[22] M. Stonebraker, B. Rubenstein, and A. Guttman. Application of Abstract Data Types and Abstract Indices to CAD Data Bases. In *ACM/IEEE Conf. on Engineering Design Applications*, pages 107–113, 1983.

[23] M. Stonebraker, B. Rubenstein, and A. Guttman. Inclusion of New Types in Relational Database Systems. In *Int. Conference on Data Engineering*, pages 262–269, 1986.

[24] A. U. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. Snodgrass. *Temporal Databases: Theory, Design, and Implementation*. The Benjamin/Cummings Publishing Company, 1993.

[25] M. Worboys. A Unified Model for Spatial and Temporal Information. *The Computer Journal*, 37(1):25–34, 1994.

[26] T. S. Yeh and B. Cambray. Time as a Geometric Dimension for Modeling the Evolution of Entities: A 3D Approach. In *Int. Conf. on Integrating GIS and Environmental Modeling*, 1993.

[27] T. S. Yeh and B. Cambray. Modeling Highly Variable Spatio-Temporal Data. In *6th AustraliAsian Database Conf.*, pages 221–230, 1995.