# 5 Spatio-temporal Models and Languages: An Approach Based on Constraints

Stéphane Grumbach[1], Manolis Koubarakis[2],
Philippe Rigaux[3], Michel Scholl[1], and Spiros Skiadopoulos[4]

[1] INRIA, Rocquencourt, France
[2] Technical University of Crete, Chania, Crete, Greece
[3] CNAM, Paris, France
[4] National Technical University of Athens, Greece

## 5.1 Introduction

The introduction of spatio-temporal information in database systems presents us with an important data modelling challenge: the design of data models general and powerful enough to handle conventional thematic data, purely temporal or spatial concepts and spatio-temporal concepts.

General purpose DBMS (e.g., relational) are not appropriate for storing and manipulating spatio-temporal data because of the complex structure of temporal and geometric information and the intricate temporal and spatial relationships among sets of related objects. Moreover, the costly operations involved in temporal and spatial object management seem at first glance to prevent the logical-level approach based on simple data structures (e.g., relations) manipulated through a limited set of simple operations (e.g., relational algebra).

In the temporal world the main research trend has therefore been to introduce a variety of temporal data models with *new temporal operations*, and then worry about the semantics and appropriateness of these operations [45,44]. Similarly, in the spatial world the main approach has been to introduce extensions of conventional database models with abstract spatial data types encapsulating geometric structures and operations (see for example [39,35,40,19,20]). Chapter 4 of this book follows this school of thought for the modeling and querying of spatio-temporal information. Similarly, in the commercial world, DBMS such as Oracle [21] and Illustra [47] provide in their latest version separate modules devoted to temporal and spatial data.

In this chapter we take an approach that differs significantly from the trends outlined above. We notice that the abstract data type approach *lacks uniformity* for the representation of data. To avoid this, we adopt the constraint data model, first introduced by Kanellakis, Kuper, and Revesz [25,33], and show that it is a very successful paradigm for the representation of spatio-temporal data in a *unified* framework. The constraint data model allows a uniform representation of all kinds of information present in spatio-temporal applications, and supports *declarative query languages* well-suited for complex spatio-temporal queries. Contrary to the temporal and spatial data models mentioned above, it also enables the straightforward modeling of *indefinite information*, a feature particularly useful in many spatio-temporal applications.

T. Sellis et al. (Eds.): Spatio-temporal Databases, LNCS 2520, pp. 177–201, 2003.

The basic idea of the constraint data model is to represent temporal and spatial objects as infinite collections of points satisfying first-order formulae. For example, an interval is defined by the conjunction of two order constraints. Similarly, a non-convex polygon, which is the intersection of a set of half-planes, is defined by the conjunction of the inequalities defining each half-plane. Finally, a non-convex polygon is defined by the union (logical disjunction) of a set of convex polygons.

In this chapter we use first-order formulae with linear (and in some cases polynomial) constraints for the definition of spatio-temporal objects. Linear constraints over the rational numbers have been shown to be a flexible and powerful way to represent many kinds of temporal data e.g., absolute, relative, periodic and indefinite [24,1,26,27,37,46]. Utilising the power of linear constraints, the user can *avoid worrying about the nasty details of special temporal operations* (as in many of the models in [45,44]), and *instead use standard relational algebra* for expressing temporal queries. There is also the added benefit of having a uniform way to represent infinite and indefinite information [24,26] something which is not possible in other temporal models.

Similarly, linear constraints over rational numbers have been shown to be appropriate for the symbolic representation of spatial and spatio-temporal data [16,32,33]. In this paradigm spatio-temporal objects can be seen as infinite sets of points at the abstract level, and can be represented by quantifier-free formulas with linear constraints at the symbolic level. This approach contrasts with the representation of objects by their boundary, which leads to cumbersome data models with ad-hoc operations, and no standard emerging. The fundamental benefits of the constraint approach is a *uniform representation* of any kind of spatio-temporal object, *no limitation in the dimension*, and *independence from the physical level* and its algorithms (i.e., potential for query optimization). Most importantly, it is also possible to manipulate spatio-temporal objects through the *standard languages* of relational calculus and algebra.

This chapter is organized as follows. In the following section, we introduce the original constraint database model of [25] and present examples of representing and querying definite spatio-temporal information. In Section 5.3 we extend this model to account for indefinite spatio-temporal information following the proposal of [28,27]. Section 5.4 points out some shortcomings of the original constraint database model when used to represent spatio-temporal data. Then it introduces the data model and algebra of the system DEDALE that has been designed to overcome these shortcomings. Section 5.5 presents the user query language for DEDALE and shows it in action in a real-life spatio-temporal application. Finally, Section 5.6 summarizes the chapter and discusses related research.

## 5.2 Representing Spatio-temporal Information Using Constraints

We introduce here the framework of *constraint databases* [25,33] which extends the classical paradigm of the relational model, and demonstrate that spatio-temporal data can be represented gracefully using constraints.

In this framework spatio-temporal data are modeled as infinite sets in the rational space. For example, a convex interval on the rational line is seen as an infinite set of points bounded by two endpoints. A polygon on the plane is seen as the infinite set of points of $\mathbb{Q}^2$ inside its frontier, and a 3-dimensional pyramid is seen as the infinite set of points of $\mathbb{Q}^3$ inside its facets. Similarly, the trajectory of a moving object is seen as an infinite set of points representing the various positions of the object during successive intervals of time (see Example 1).

Following the terminology of Chapter 4 of this book, our *abstract model* of spatial data consists of infinite relations over the universe of the rational numbers $\mathbb{Q}$. These infinite relations can only be described and manipulated through a finite representation. Following the trends of constraint databases [25], we use first-order logic to represent the relations of interest, and define a *symbolic level* of representation (a discrete model in the terminology of Chapter 4). We distinguish clearly between the *intensional* representation of a relation at the symbolic level, and its *extensional* interpretation at the abstract level.

We consider linear constraints in the first-order language $\mathcal{L} = \{\leqslant, +\} \cup \mathbb{Q}$ over the structure $\mathcal{Q} = \langle \mathbb{Q}, \leqslant, +, (q)_{q\in\mathbb{Q}} \rangle$ of the linearly ordered set of the rational numbers with rational constants and addition. Constraints are linear equations and inequalities of the form: $\sum_{i=1}^{p} a_i\, x_i \Theta a_0$, where $\Theta$ is a predicate among $=$ or $\leqslant$, the $x_i$'s denote variables and the $a_i$'s are integer constants. Note that rational constants can always be avoided in linear equations and inequalities. The multiplication symbol is used as an abbreviation, $a_i x_i$ stands for $x_i + \cdots + x_i$ ($a_i$ times).

Let $\sigma = \{R_1, ..., R_n\}$ be a database schema such that $\mathcal{L} \cap \sigma = \varnothing$, where $R_1, ..., R_n$ are relation symbols. We distinguish between *logical predicates* (e.g., $=, \leqslant$) in $\mathcal{L}$ and *relations* in $\sigma$. The basic concept of our abstract model is the linear constraint relation which is defined as follows.

**Definition 1.** Let $S \subseteq \mathbb{Q}^k$ be a $k$-ary relation. The relation $S$ is a *linear constraint relation* if there exists a formula $\varphi(x_1, ..., x_k)$ in $\mathcal{L}$ with $k$ distinct free variables $x_1, ..., x_k$ such that:

$$\mathcal{Q} \models \forall x_1 \cdots x_k (S(x_1, ..., x_k) \leftrightarrow \varphi(x_1, ..., x_k))$$

Formula $\varphi$ is called a *representation* of $S$.

We denote by *LCR($\mathbb{Q}^k$)* the class of linear constraint relations over $\mathbb{Q}^k$. This class constitutes a rather drastic restriction over the class of all infinite relations over $\mathbb{Q}^k$. The restriction is necessary to ensure reasonable query complexity and is sufficient to encompass spatial data in computational geometry, GIS, etc. as it has already been widely demonstrated in the literature [33].

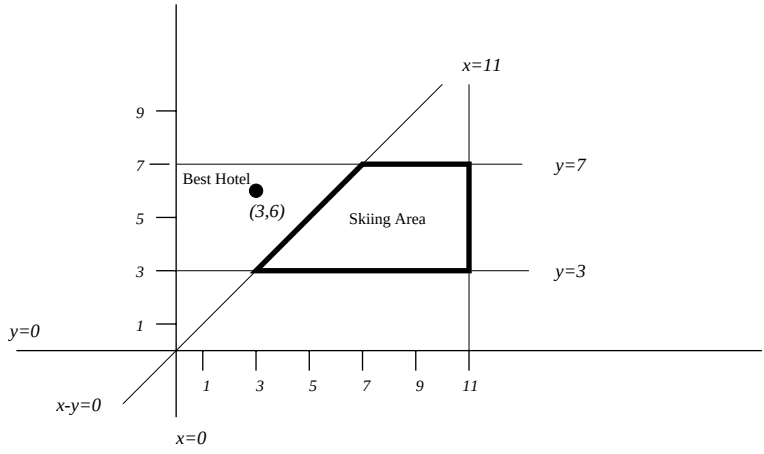


**Fig. 5.1.** A very simplified view of a ski resort

The basic concepts of our symbolic representation are the generalized tuple and the generalized relation (originally introduced in [25]).

**Definition 2.** A *generalized tuple* is a conjunction of linear constraints from the first-order language $\mathcal{L}$. A *generalized relation* is a finite set of generalized tuples (i.e., a generalized relation corresponds to a formula of $\mathcal{L}$ in disjunctive normal form or DNF).

Thus our symbolic representation offers a finite representation for infinite sets of points in $d$-dimensional space $\mathbb{Q}^d$ with no limitation on $d$.

The concept of a linear constraint relation defined above can be extended to cover relations that combine an uninterpreted domain $D$ (used for the modelling of thematic data) with the interpreted one $\mathbb{Q}$. The concept extends easily to such relations, with representation formulae in $\mathcal{L} \cup D$, with two sorts of constraints, (i) equality constraints over objects of $D$, and linear constraints over objects of $\mathbb{Q}$. We will denote by $LCR(D, \mathbb{Q}^2)$, the set of linear constraint relations over $D \times \mathbb{Q}^2$. The concepts of our symbolic representation (generalized tuple and relation) can be similarly extended with equality and inequality constraints over the domain $D$.

Let us now illustrate the use of linear constraint relations in a real life application provided by the French laboratory LAMA, Grenoble [8].

*Example 1.* We consider a ski resort and study the spatio-temporal behavior of several types of actors. The resort is known as a set of buildings and areas with well-defined utilities, e.g. hotels, night clubs, various kinds of stores and of course the skiing area. The typical behavior of human actors (tourists for instance) is statistically modeled with respect to their socio-economical category (age, income, nationality, etc.) and represented as a sequence of activities and positions during a typical vacation day. For instance the tourist category will be described as the following succession of activities: sleeping, walking, skiing,

eating, reading, watching movies, etc., each activity being associated with a time interval. The trajectory of each tourist is described by partitioning the day in time slices and associating with each slice the position(s) where a representative of the category is likely to be found during this time slice. The ultimate goal of the application is to improve the organization of the ski resort by detecting places where no one ever goes, equipment which is underutilized, categories which share the same behavior, spatial distribution of tourists in the resort with respect to time and so on.

The following generalized relation *People* represents the known information about the activities of tourist John from midnight ($t = 0$) to noon ($t = 12$) of a typical day. The location of John's hotel together with other geometric information is depicted in Figure 5.1). We have used simple linear constraints in this example although the full power of the language *caL* is available.

People

| Name | Category | Activity | Trajectory |
|---|---|---|---|
| John | Tourist | Sleeping | $0 \leqslant t < 8 \wedge x = 3 \wedge y = 6$ |
| John | Tourist | Eating | $8 \leqslant t < 9 \wedge x = 3 \wedge y = 6$ |
| John | Tourist | Skiing | $9 \leqslant t < 12 \wedge x - y \geqslant 0 \wedge y \geqslant 3 \wedge y \leqslant 7 \wedge x \leqslant 11$ |

The column to notice is $Trajectory$ (the rest of the columns are as in standard relational databases). The attribute values for column $Trajectory$ form an infinite set of triples $(t, x, y)$ which is represented in a finite way by linear constraints. $Trajectory$ essentially represents three attributes in the sense of the relational model (time plus two coordinates), each of them with an infinite set of values. We choose not to show these attributes explicitly and refer to them by using the corresponding variables $t, x$ and $y$. We will follow the same practice in the rest of the paper. When we give examples of generalized relations we often blur the distinction between the columns or attributes, and the corresponding variable names.

The above definitions of linear constraint relations and generalized relations are easily extended to definitions for *linear constraint databases* and *generalized databases.* The details are omitted for brevity.

### 5.2.1 An Algebra for Relations with Constraints

Spatio-temporal data represented by generalized relations can be queried using first-order logic (relational calculus) or relational algebra [25]. In this section we will give detailed definitions of relational algebra over linear constraint relations. The definitions for relational calculus are as in the standard relational case and are omitted. Relational calculus and algebra are equivalent over linear constraint relations (this has been shown in [18,28,27,36]).

The algebra for generalized relations consists of *union*, $\cup$, *cartesian product*, $\times$, *difference*, $-$, *selection*, $\sigma_F$, where $F$ is an atomic constraint, and *projection*, $\pi$. These operations are defined as follows.

Let $R_1$ and $R_2$ be two relations, and respectively $e_1$ and $e_2$ be sets of generalized tuples defining them.

1. $R_1 \cup R_2 = e_1 \cup e_2$.
2. $R_1 \times R_2 = \{t_1 \wedge t_2 \mid t_1 \in e_1, t_2 \in e_2\}$.
3. $R_1 - R_2 = \{t_1 \wedge t_2 \mid t_1 \in e_1, t_2 \in (e_2)^c\}$,
   where $e^c$ is the set of tuples or disjuncts of a DNF formula corresponding to $\neg e$.
4. $\sigma_F(R) = R \times \{F\}$.
5. $\pi_{\overline{x}}\ R_1 = \{\pi_{\overline{x}}\ t \mid t \in e_1\}$,
   where

$$\pi_{\overline{x}}\ t = \bigwedge_{1 \leqslant k \leqslant K, 1 \leqslant \ell \leqslant L} b^k \overline{x} - b_0^k \leqslant a_0^\ell - a^\ell \overline{x} \ \wedge \bigwedge_{1 \leqslant i \leqslant I} c^i \overline{x} \leqslant c_0^i.$$

   is given by the Fourier-Motzkin Elimination method [42] from a tuple $t$ defining a polyhedron $P(\overline{x}, y) \subseteq \mathbb{Q}^{n+1}$ described by the inequalities (once the coefficients of $y$ have been normalized):

$$\begin{cases} a^\ell \overline{x} + y \leqslant a_0^\ell & \text{for } \ell = 1, ..., L \\ b^k \overline{x} - y \leqslant b_0^k & \text{for } k = 1, ..., K \\ c^i \overline{x} \leqslant c_0^i & \text{for } i = 1, ..., I \end{cases}$$

   where $\overline{x}$ ranges over $\mathbb{Q}^n$, and $y$ over $\mathbb{Q}$.

Using the above operations *join* can easily be defined using cartesian product and selection. Also, *intersection* of two generalized relations over a common set of attributes is easily seen to be equal to the join of these relations.

The semantics of the symbolic operators applied to sets of generalized tuples simulates relational operators applied to infinite relations, and provides a correct mathematical representation of the result that complies with the constraint representation. For selection and cross product, this is done in a somehow lazy way, by just concatenating the input(s). The result might be inconsistent or redundant: a semantic evaluation, denoted *simplification*, must be carried out at some step of the query execution process in order to eliminate redundancies and to detect inconsistencies. The algorithms and complexities for the constraints manipulation can be found in [15,16]. For the purposes of our discussion let us just consider an example.

*Example 2.* Let us consider the database of Example 1 and the query "Where is John between 10 and 12?". In relational calculus, this query can be expressed by the formula

$$name, x, y: \ (\exists t)(People(name, t, x, y) \wedge name = John \wedge 10 \leqslant t < 12)$$

and its answer is the following relation:

| Name | Place |
|---|---|
| John | $x - y \geqslant 0 \wedge y \geqslant 3 \wedge y \leqslant 7 \wedge x \leqslant 11$ |

In the algebra for generalized relations, the same query can be expressed as follows:

$$\pi_{Name,x,y}(\sigma_{Name=John \wedge 10 \leqslant t < 12}(People))$$

To evaluate this algebraic query we first apply selection to each tuple of the relation *People*. The result of this operation is the tuple

$$(John,\ Tourist,\ Ski,\ 10 \leqslant t < 12 \wedge x - y \geqslant 0 \wedge y \geqslant 3 \wedge y \leqslant 7 \wedge x \leqslant 11)$$

This tuple comes from the third tuple of this relation (the first two tuples do not survive the selection because they contain constraints that contradict the selection condition $12 \leqslant t < 14$). Finally, the projection operation is executed (it is straightforward for this example!) to arrive at the result.

## 5.3 Indefinite Information in Spatio-temporal Databases

The ideas of the previous section can be extended to accommodate indefinite information [26,27,28,29,30,31,32]. The resulting *indefinite constraint database scheme* is very powerful and it can be used to model many new spatio-temporal applications. This scheme is also important from the point of view of Artificial Intelligence because it essentially unifies the representational capabilities of *constraint networks* [12] with these of relational databases.

To motivate the need to represent indefinite information consider Figure 5.2 which exhibits a situation that might be represented in the database of the Ministry of the Environment and Natural Resources of some European country. We assume that there is some species inhabiting the rectangular area $A$. Now assume that there is some form of atmospheric pollution which has been generated due to some industrial accident at point $(5, 5)$. The extent of the pollution is not known precisely at this time. All we know is that the minimum area polluted is given by rectangle $B$ and the maximum by rectangle $C$.[1]

In this example our information about the species occupying area $A$ is *definite* (i.e., we know *precisely* the location and extent of the species habitat). Unfortunately we have *indefinite* or *imprecise* information about the polluted area. Due to our lack of precise information about the polluted area, the situation in the real world is compatible with many possibilities or *possible worlds* as they have been called by philosophers and logicians [22]. In one possible world the polluted area might be enclosed exactly by rectangle $B$. In another the polluted area might be enclosed by the rectangle defined by points $(1, 1)$ and $(7, 7)$. In fact, there is an infinite number of such possible worlds and in each one of them the polluted area is defined by a rectangle enclosing rectangle $B$ and enclosed by rectangle $C$.

At the abstract level, each one of these possible worlds corresponds to two infinite sets of points defining the species habitat and the polluted area. The

[1] We do not address here the question whether a rectangle is an appropriate geometric means for capturing this kind of data.

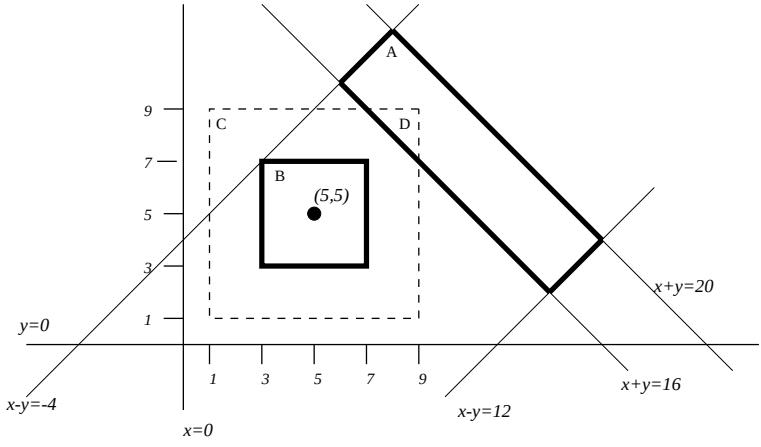


**Fig. 5.2.** Species (A) and pollution (B, C)

ideas of Section 5.2 now imply that, at the symbolic level, each possible world can be represented by a generalized database. For example, if we assume that the polluted area is enclosed exactly by rectangle $B$, the information in the above example can be captured by the following generalized relations:

*Species*

| *Area* |
|---|
| $x + y \leqslant 20 \wedge x + y \geqslant 16 \wedge x - y \leqslant 12 \wedge x - y \geqslant -4$ |

*Polluted*

| *Area* |
|---|
| $x \geqslant 3 \wedge x \leqslant 7 \wedge y \geqslant 3 \wedge y \leqslant 7$ |

The above example motivates us to consider *sets of possible worlds* i.e., *sets of linear constraint relations* as the appropriate formal concept for the representation of indefinite spatio-temporal information in our abstract model. This is captured by the following definitions.

**Definition 3.** A *possible world* is a linear constraint relation.

Possible worlds can be used for the representation of definite information as explained above. When we have indefinite information, the concept of indefinite linear constraint relation is required.

**Definition 4.** Let $\mathbf{S} \subseteq \wp(\mathbb{Q}^k)$ be a set of possible worlds (the notation $\wp(\cdot)$ denotes powerset). The set of possible worlds $\mathbf{S}$ is an *indefinite linear constraint relation* if there exist formulae $CS(\omega_1, ..., \omega_n)$ and $\varphi(x_1, ..., x_k, \omega_1, ..., \omega_n)$ in $\mathcal{L}$ with distinct free variables $\omega_1, ..., \omega_n$ and $x_1, ..., x_k$ such that:

$$\mathbf{S} = \{\ W \subseteq \mathbb{Q}^k :\ (x_0^1, \ldots, x_0^k) \in W \text{ iff}$$
$$\mathcal{Q} \models \exists \omega_1 \cdots \omega_n (CS(\omega_1, ..., \omega_n) \wedge \varphi(x_0^1, \ldots, x_0^k, \omega_1, ..., \omega_n))\ \}$$

We denote by *ILCR*($\mathbb{Q}^k$) the set of indefinite linear constraint relations over $\mathbb{Q}^k$. As in the previous section we also consider relations that combine the uninterpreted domain $D$ with the interpreted one $\mathbb{Q}$. In the past indefinite relations over an uninterpreted domain had been studied in the literature on *null values* ([23,14] are the most outstanding works in this area).

It is straightforward to extend the above definition to cover the concept of indefinite linear constraint databases as sets of indefinite linear constraint relations [28,27]. The details are omitted for brevity.

At the symbolic level, indefinite linear constraint relations (or databases) will be represented by indefinite generalized relations (or databases). These concepts are defined below.

**Definition 5.** An *indefinite generalized relation* is a set of generalized tuples (as defined in Definition 2). These tuples are also allowed to contain Skolem constants. *Skolem constants*, usually denoted by $\omega_1, ..., \omega_n$, are essentially equivalent to the existentially quantified variables $\omega_1, ..., \omega_n$ of the formula

$$\exists \omega_1 \cdots \omega_n (CS(\omega_1, ..., \omega_n) \wedge \varphi(x_0^1, \ldots, x_0^k, \omega_1, ..., \omega_n))$$

of Definition 4.

**Definition 6.** An *indefinite generalized database* is a set of indefinite generalized relations with an associated *constraint store* $CS(\omega_1, ..., \omega_n)$. The constraint store is a quantifier free formula of $\mathcal{L}$ which is used to constrain the Skolem constants of all generalized relations.

*Example 3.* The following indefinite generalized database represents the information in Figure 5.2.

Species

| *Area* |
|---|
| $x + y \leqslant 20 \wedge x + y \geqslant 16 \wedge x - y \leqslant 12 \wedge x - y \geqslant -4$ |

Polluted

| *Area* |
|---|
| $5 - \omega \leqslant x \wedge x \leqslant 5 + \omega \wedge 5 - \omega \leqslant y \wedge y \leqslant 5 + \omega$ |

$$CS(\omega): \quad \omega \geqslant 2 \wedge \omega \leqslant 4$$

The following example modelled after [43] deals with moving objects.

*Example 4.* Let us consider an object moving on a straight line in $\mathbb{Q}^2$ with motion vector $d = (d_1, d_2)$ and speed $v$. Let us also assume that its initial position at time $t_0$ is $(x(t_0), y(t_0))$. The position $(x(t), y(t))$ of the object at future time $t$ can be computed using the following equations:

$$x(t) = x(t_0) + v(t - t_0)d_1 \text{ and } y(t) = y(t_0) + v(t - t_0)d_2$$

In this example we go beyond linear constraints and consider *indefinite polynomial constraint databases*. For the definite case, polynomial constraint databases have been studied in various papers [25,33]. We do not give detailed definitions for this case, since the required definitions are similar to the ones above, and follow immediately from the general scheme of indefinite constraint databases given in [27].

We consider a concrete example of moving objects and assume that we have an indefinite polynomial constraint database containing information about the moving object $Car1$:

InitPos

| Id | Time | Xcord | Ycord |
|---|---|---|---|
| $Car1$ | 0 | 1 | 1 |

Vector

| Id | Xval | Yval |
|---|---|---|
| $Car1$ | 2 | 1 |

Speed

| Id | Value |
|---|---|
| $Car1$ | $\omega$ |

$$CS(\omega): \ 40 \leqslant \omega \leqslant 50$$

$$FuturePos(o, t, x_0 + v(t - t_0)d_1, y_0 + v(t - t_0)d_2) \leftarrow InitPos(o, t_0, x_0, y_0), \\ Vector(o, d_1, d_2), \\ Speed(o, v)$$

The only generalized relation with truly indefinite information is *Speed*. The speed of $Car1$ is not known precisely: all we know is that it is between 40 and 50 miles per time unit. The future position of any object is computed using the rule which defines the generalized relation $FuturePos$ with schema $(Id, \ Time, \ X, \ Y)$. We do not give a detailed semantics for such rules since they follow immediately from relevant literature [25,28].

In the next section, we will pose queries to the above database to compute the position of object $Car1$ at times other than 0.

### 5.3.1 Querying Indefinite Information

An algebra and calculus for indefinite constraint databases has been defined in [26,27,28,29,30]. The calculus is an extension of standard relational calculus with the *modal* operators $\Diamond$ and $\Box$ for expressing *possibility* and *certainty* queries. The motivation for introducing modal operators for the querying of databases with indefinite information has been given in many earlier papers [23,14].

This modal calculus and the corresponding algebra are equivalent over indefinite linear constraint databases [27,28]. We omit detailed definitions for the

calculus and instead concentrate on the algebra (but we show examples of calculus queries below).

The algebra for indefinite generalized relations extends the one discussed in Section 5.2 as follows:

1. The definitions of union, cartesian product, difference, selection and projection remain unchanged. These operations do not interfere with Skolem constants or the constraint store.
2. If only these operations are contained in a query, the answer relation might contain Skolem constants. Thus for an answer relation to be meaningful it must be accompanied by a *constraint store* which gives meaning to the Skolem constants.
   If $CS(\omega_1, ..., \omega_n)$ is the constraint store of the input database, the constraint store of the answer relation is $\pi_{\omega_{i1},...,\omega_{im}}(CS(\omega_1, ..., \omega_n))$ where $\omega_{i1}, ..., \omega_{im}$ are these Skolem constants among $\omega_1, ..., \omega_n$ that appear in the answer relation.
3. Two new *modal* operators $POSS$ (for *possibility*) and $CERT$ (for *certainty*) are introduced. These operators correspond to the natural language expressions "possibly" and "certainly" that might appear in a query over a database with indefinite information.
   $POSS$ and $CERT$ take into account the Skolem constants and the constraints in the constraint store. Applying these operators on an input indefinite relation, results in a *definite relation*. In other words, the information in the database might be indefinite but we can *definitely* compute what is possible (or certain) given the information in the database.

Before we proceed to define the new operators $POSS$ and $CERT$, let us give an example of a query which does not involve these operators.

*Example 5.* Consider the database of Example 3 and the query "Compute the intersection of the area inhabited by the species with the polluted area".

In the calculus for indefinite linear constraint relations this query can be expressed as follows:

$$x, y: \ Species(x, y) \wedge Polluted(x, y)$$

In the algebra for indefinite generalized relations the query can be expressed by $Species \cap Polluted$. Evaluating this expression is straightforward and gives the following result:

SpeciesInDanger

| $Area$ |
|---|
| $x + y \leqslant 20 \wedge x + y \geqslant 16 \wedge x - y \leqslant 12 \wedge x - y \geqslant -4 \wedge$ $5 - \omega \leqslant x \wedge x \leqslant 5 + \omega \wedge 5 - \omega \leqslant y \wedge y \leqslant 5 + \omega$ |

$$CS(\omega): \ \ \omega \geqslant 2 \wedge \omega \leqslant 4$$

Notice that the Skolem constant $\omega$ is contained in the constraints of the answer relation. Thus for the answer relation to be meaningful we include the projection of $CS(\omega)$ on $\omega$.

An answer such as the one above might be hard to understand even with the provision of formula $CS(\omega)$. Towards the end of this section we propose ways around this problem.

Let us now define operation $POSS$. Let $R$ be an indefinite linear constraint relation, $e$ the indefinite generalized relation representing $R$, and $CS(\omega_1, ..., \omega_n)$ the constraint store associated with $R$. Then $POSS(R)$ is a new *linear constraint relation* over the same schema consisting of all tuples in *any* of the possible worlds of $R$.

A more operational definition of $POSS(R)$ is as follows:

$$POSS(R) = \{\ \pi_{var(t)}(t \wedge CS(\omega_1, ..., \omega_n)) :\ t \in e \text{ and}$$

$$var(t) \text{ is the set of variables of tuple } t\ \}$$

In other words, each tuple in the generalized relation representing $POSS(R)$ is obtained by eliminating all Skolem constants from the conjunction of constraints $CS(\omega_1, ..., \omega_n) \wedge t$ where $t$ is a generalized tuple of the symbolic representation of $R$.

The following queries use operation $POSS$.

*Example 6.* Consider the database of Example 3 and the query "Compute the area inhabited by the species which has possibly been polluted by the accident".

In the calculus for indefinite linear constraint relations this query can be expressed as follows:

$$x, y :\ \Diamond(Species(x, y) \wedge Polluted(x, y))$$

In the algebra for indefinite generalized relations the query can be expressed by $POSS(Species \cap Polluted)$. Evaluating this expression involves computing the intersection $Species \cap Polluted$ as above and then calculating $\pi_{x,y}(t)$ for all tuples $t$ of $Species \cap Polluted$.

The calculation of

$$\pi_{x,y}(x + y \leqslant 20 \wedge x + y \geqslant 16 \wedge x - y \leqslant 12 \wedge x - y \geqslant -4 \wedge 5 - \omega \leqslant x \wedge$$

$$x \leqslant 5 + \omega \wedge 5 - \omega \leqslant y \wedge y \leqslant 5 + \omega \wedge$$

$$\omega \geqslant 2 \wedge \omega \leqslant 4)$$

results in

$$x \leqslant 9 \wedge y \leqslant 9 \wedge x + y \geqslant 16$$

and corresponds to the area $D$ of Figure 5.2. Thus the answer to the query is the following:

SpeciesPossiblyInDanger

| *Area* |
|---|
| $x \leqslant 9 \wedge y \leqslant 9 \wedge x + y \geqslant 16$ |

$$CS(\omega): \quad true$$

Notice that the returned constraint store is empty (contains only the trivial constraint *true*) because the answer relation is definite.

*Example 7.* Consider the indefinite constraint database of Example 4 and the query "What are the possible future positions of object $Car1$ at time 5".

In the calculus for indefinite linear constraint relations this query can be expressed as follows:

$$x, y: \ \Diamond FuturePos(Car1, 5, x, y)$$

In the algebra for indefinite generalized relations this query can be expressed by

$$\pi_{X,Y}(POSS(\sigma_{Id=Car1 \wedge Time=5}(FuturePos)))$$

and has the following answer:

PosAt5

| *Area* |
|---|
| $401 \leqslant x \leqslant 501 \wedge y = \frac{1}{2}x + \frac{1}{2}$ |

$$CS: \ true$$

Because the database does not contain definite information about the speed of $Car1$, a line segment in $\mathbb{Q}^2$ is returned as the answer to the query.

Let us now define operation $CERT$. Let $R$ be an indefinite linear constraint relation, $e$ the indefinite generalized relation representing $R$, and $CS(\omega_1, ..., \omega_n)$ the constraint store associated with $R$. Then $CERT(R)$ is a new *linear constraint relation* over the same schema consisting of only these tuples that are contained in *every* possible world of $R$.

A more operational definition of $CERT$ is as follows:

$$CERT(R) = \{\ \pi_{var(t)}(CS(\omega_1, ..., \omega_n) \wedge t): \ t \in e^c \text{ and}$$

$$var(t) \text{ is the set of variables of tuple } t\ \}^c$$

where the application of the operator $^c$ can be understood as follows. If $A$ is a set of generalized tuples then $A^c$ is the set of tuples or disjuncts of a DNF formula corresponding to formula $\neg\psi$ where $\psi$ is the formula in DNF corresponding to $A$.

As witnessed by its definition, $CERT$ is a very expensive operation [30,31,32]. The following example uses operation $CERT$.

*Example 8.* Let us consider the database of Example 3 and the query "Compute the area occupied by the species which has certainly been polluted by the accident".

In the calculus for indefinite linear constraint relations this query can be expressed as follows:

$$x, y : \ \Box(Species(x, y) \wedge Polluted(x, y))$$

In the algebra for indefinite generalized relations the query can be expressed by $CERT(Species \cap Polluted)$. If we evaluate this query, we get the empty relation (as it might be expected by examining Figure 5.2).

The queries of Example 6 and 8 should now be compared with the query of Example 5. We have already mentioned that the answer to the query of Example 5 is rather hard to understand even with the provision of the constraint $CS(\omega)$. The operators $POSS$ and $CERT$ can assist in comprehending the answer in such cases because they provide *upper* and *lower bounds* to the set of tuples contained in the answer to the original query. With the provision of a graphical user interface, the user can then be shown the geometric objects corresponding to these upper and lower bounds. This will greatly enhance the comprehension of the returned answers.

## 5.4 Beyond Flat Constraint Relations: The DEDALE Approach

As we have shown in Sections 5.2 and 5.3, the original constraint database model of [25] (and its extension to the indefinite case [28]) gives us a powerful framework for representing and querying spatio-temporal data. However, it has been observed [3,15,16,34] that this version of the constraint database model has two shortcomings:

- Constraint query languages like the ones of Sections 5.2 and 5.3 do not always offer natural means for expressing spatio-temporal queries. This shortcoming is due to the fact that spatio-temporal objects must be "broken" into component tuples in the original constraint database model (for instance, the relation in Example 1 has 3 tuples for the same object).
  This observation has led to the development of constraint database models based on non-first normal form relations [2]. Our own data model for the system DEDALE [15,16] is discussed below. Similar models have been developed independently by [3,34].
- Often the chosen constraint language is not expressive enough. For example, the language $\mathcal{L}$ of linear constraints used in Sections 5.2 and 5.3 cannot express several interesting spatial functions e.g., distance or convex hull [3,15,16].
  This problem can be solved in two ways. We can go to a more powerful constraint language (e.g., polynomial constraints, as we did in Example 4)[2]

[2] But this is not always satisfactory because all the implementation advantages of linear constraints can be lost [3].

or we can carefully introduce appropriate additional primitives in the query language. The latter approach has been followed by [3,15,16,34].

Let us now present the data model of DEDALE which is a carefully designed formalism for addressing the above considerations [15,16]. This data model extends the original constraint-database model presented in Section 5.2 thus it does *not* support indefinite information. The addition of functionality for the support of indefinite information in DEDALE is currently an open question. This section concentrates only on the data model and languages of DEDALE while the system itself is discussed in Chapter 7.

In the DEDALE data model spatio-temporal data is treated as (possibly infinite) sets of points in $\mathbb{Q}^d$ space with no limitation on the dimension $d$. These sets are then used as values in tuples as it is the case in nested relational models [2]. The interesting thing to notice here is that *nesting can be limited to one level* (this suffices for spatio-temporal data). In addition, we allow to construct sets of points in the uninterpreted domain as well as in the rational domain, in order in particular to represent non-geometric time-evolving attributes.

We next introduce the data types of our model. In contrast to the approach of Chapter 4 our datatypes only distinguish between spatio-temporal and thematic data (i.e., we do *not* have a special data for each kind of spatio-temporal object).

We assume the existence of two *atomic types* $\mathcal{Q}$ and $U$ called the *rational* and *uninterpreted type* respectively. The domains of these types are $\mathbb{Q}$ and $D$. In practice we might need many uninterpreted types to cater for different kinds of thematic data; we ignore this issue in the rest of this paper.

We also have the following complex types: *set type, tuple type* and *relation type*. These types are defined as follows:

**Definition 7.** 1. If $A_1, \ldots, A_{k_0}, A_{l_1}, \ldots, A_{l_i}$ are attribute names and $k_0, k_1, \ldots, k_i$ are positive natural numbers then

$$\{[A_1 : U, \ldots, A_{k_0} : U, A_{l_1} : \mathcal{Q}^{k_1}, \ldots, A_{l_i} : \mathcal{Q}^{k_i}]\}$$

denotes a *set type* with domain

$$LCR(D^{k_0}, \mathbb{Q}^{k_1}, \ldots, \mathbb{Q}^{k_i}).$$

2. If $T_1, ..., T_n$ are atomic or set types, and $A_1, ..., A_n$ are attribute names then

$$[A_1 : T_1, ..., A_n : T_n]$$

is a *tuple type* with domain

$$dom([A_1 : T_1, ..., A_n : T_n]) = \{[A_1 : a_1, ..., A_n : a_n] \mid a_i \in dom(T_i)\}.$$

3. If $T$ is a tuple type then $\{T\}$ is a *relation type* with domain

$$dom(\{T\}) \ = \ \wp_f(dom(T))$$

where $\wp_f(S)$ denotes the set of finite subsets of $S$.

A *relation schema* is a relation type. A *database schema* is a finite collection of relation types. An *instance* of a relation schema is defined as usual. In the sequel of this chapter the word *relation* is used for an object of relation type, and we distinguish *relations* from *sets* of set type.

*Example 9.* Let us consider again the ski-resort application of Example 1. The following is a relation schema for the actors of the application (considered as moving objects with time-varying activities).

$$
\begin{aligned}
People = \{[\ & \text{Name} : string, \\
& \text{Category} : string, \\
& \text{Activity} : \{[\text{Name} : string, \text{Time} : \mathcal{Q}]\}, \\
& \text{Trajectory} : \{[\text{Space} : \mathcal{Q}^2, \text{Time} : \mathcal{Q}]\} \\
]\}&
\end{aligned}
$$

The following table shows how the instance of relation *People* from Example 1 can now be expressed in our nested model.

| Name | Category | Activity | Trajectory |
|---|---|---|---|
| John | Tourist | (name=‘‘Sleeping’’ $\wedge\ 0 \leqslant t < 8$) $\vee$ (name=‘‘Eating’’ $\wedge\ 8 \leqslant t < 9$) $\vee$ (*name=“Skiing”* $\wedge\ 9 \leqslant t < 12$) | $(x = 3 \wedge y = 6 \wedge 0 \leqslant t < 9) \vee (x - y \geqslant 0 \wedge\ y \geqslant 3 \wedge y \leqslant 7 \wedge x \leqslant 11 \wedge\ 9 \leqslant t < 12)$ |

In the above example the atomic type *string* is used as an uninterpreted type. Attributes *Name* and *Category* are represented as classical atomic values, while the nested attributes *Activity* and *Trajectory* are relations in respectively $LCR(D, \mathbb{Q})$ and $LCR(\mathbb{Q}^2, \mathbb{Q})$ which are represented as FO formulas. For example, in *Trajectory*, a constraint $c$ bounds either the first two coordinates (interpreted as $x$ and $y$) or the last one (interpreted as $Time$). The trajectory can be seen as a sequence of time intervals associated with the point set where the object can be found during this interval. Figure 5.3 shows graphically another example of a similar spatio-temporal object.

### 5.4.1 The DEDALE Algebra

We now consider an algebraic query language to manipulate the complex relations introduced above. The DEDALE algebra was originally defined in [15,16] and its basic operations are as follows:

- *Set operations*: *union*, $\cup$, *intersection*, $\cap$, and *set difference*, $-$, apply to pairs of inputs of the same *set* or *relation* type.
- *Selection*, $\sigma_F$, applies to inputs of *relation* or *set* type. $F$ is an atomic constraint over variables corresponding to the attributes given by name or position. This constraint is either of linear form (e.g., $4X + 3Y = 2$), or a set membership constraint (e.g., $X \in S$).

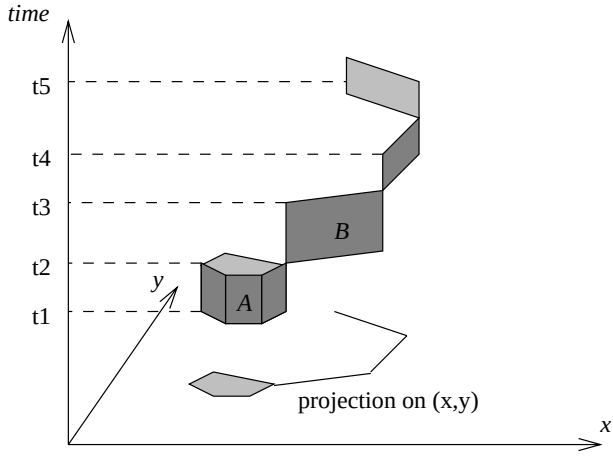


**Fig. 5.3.** A spatio-temporal object

- *Projection*, $\pi$, applies to inputs of *set* or *relation* type.
  For objects $t$ of tuple type, $t.i$ denotes the $i$'th attribute when relevant.
- *Cartesian product*, $\times$, applies to pairs of inputs of either both *set* types or both *relation* types.
- *Restructuring*, $MAP$ applies to inputs of *relation* type. If $E(X)$ is an algebraic expression of tuple type $T'$, with a tuple variable $X : T$, and $R$ is a relation of type $\{T\}$, then $MAP_{\lambda X.E(X)}(R)$ defines the relation of type $\{T'\}$, in which each tuple $t$ of $R$, has been replaced by $E(t)$.

The semantics of the above operations on inputs of *relation* type corresponds to the semantics of classical relational algebra over finite relations, while the interpretation of the operations on inputs of *set* type corresponds to the semantics of the operations on linear constraint relations as presented in Section 5.2.1.

*Example 10.* Let us consider the following *clipping query* over the relation *People* of Example 9: "Give those people who crossed the area specified by rectangle *Rect* and their associated trajectory". Assuming that *Rect* is represented by the constraints

$$\{\ 9 \leqslant x \leqslant 13,\ 5 \leqslant y \leqslant 9\ \}$$

the algebra expression for this query is:

$$MAP_{\lambda X.[X.name,(Rect \times X.trajectory)]}(People)$$

Since *Rect* and *People* are defined partly on the same variables, the expression $Rect \times X.trajectory$ is a (natural) join on $x$ and $y$. As in the relational case, its semantics is the triplets $(x, y, t)$ in the trajectory of *People* such that $(x, y)$ can be also found in *Rect*. Since both extensions are infinite, this result is represented by linear constraints.

For the instance of Example 9 the above query yields the following result:

| Name | Trajectory |
|---|---|
| John | $(x = 3 \wedge y = 6 \wedge 0 \leqslant t < 9$<br>$\wedge\ 9 \leqslant x \leqslant 13 \wedge 5 \leqslant y \leqslant 9)$<br>$\vee$<br>$(x - y \geqslant 0 \wedge y \geqslant 3 \wedge y \leqslant 7 \wedge x \leqslant 11$<br>$\wedge\ 9 \leqslant t < 12$<br>$\wedge\ 9 \leqslant x \leqslant 13 \wedge 5 \leqslant y \leqslant 9)$ |

The result features a new formula for *trajectory* which represents exactly those points with coordinates satisfying both the formula that represented the initial trajectory and the formula representing the rectangle: in other words the intersection on $x$ and $y$. Note that the first disjunct in $Trajectory$ is unsatisfiable and can be deleted. Similarly the constraints in the second disjunct can be simplified to arrive at the following relation:

| $Name$ | $Trajectory$ |
|---|---|
| John | $x \geqslant 9 \wedge x \leqslant 11 \wedge y \geqslant 5 \wedge y \leqslant 7$<br>$\wedge\ 9 \leqslant t < 12$ |

In summary, any relational algebraic expression can be applied through the $MAP$ operator to point sets: this allows to express easily intersections (spatial joins), (geometric) projections, differences, complex selections, etc. The language is abstract and general: it is uniform for alphanumerical and spatial data, and does not limit the dimension or the geometric type of objects.

The DEDALE algebra also contains the primitive operations *unnest* and *unionest*, and some other operations definable using the primitive operations [15,16]. The algebra also includes new operators for $axis, median, dist$ (for distance) and *connect?* (for connectivity). These operators have been added in order to be able to express interesting spatial queries such as convex hull, Voronoi diagrams, metric properties and topological queries. Linear constraint relations are closed over these operators and the queries expressed remain tractable (in the data complexity measure). More details can be found in [15,16].

## 5.5 The User Query Language of DEDALE

This section presents the user query language of DEDALE, along with some examples of queries. This language meets two requirements. First it hides as much as possible the underlying complexity of the data model. In particular the rules related to typing restrictions and to the nested structure are transparent to the end-user. Second the language has a direct and easy translation towards the DEDALE algebra, thereby allowing the design of an optimizer generating efficient evaluations.

### 5.5.1 The Syntax

Our language relies on an SQL-like syntax. It contains the following components:

1. Algebraic operations such as **join, union** etc. that operate on sets (i.e. spatial objects).
2. The limited set of primitives which have been introduced in the DEDALE algebra to augment its expressive power (**axis, median, dist**, and **connect?**).
3. Macro-operations which simplify the expression of the most common spatial data manipulations.

Note that macro-operations are simple shortcuts for complex algebraic expressions, while primitives may be considered as external functions with respect to the algebraic operators. The details of the language can be found in [16]. The rest of this section concentrates only on the algebraic operations and their use in a spatio-temporal application.

The query language follows the SQL syntax to construct algebraic expressions on nested attributes in the **select** clause as well as boolean expressions on nested attributes in the **where** clause. If $p_1$ and $p_2$ denote two pointset attributes (which can be either nested attributes or the results of algebraic expressions), then:

1. $p_1$ **join** $p_2$ denotes a *natural* join on the components common to $p_1$ and $p_2$.
2. $p_1$ **union** $p_2$ denotes the union of $p_1$ and $p_2$.
3. $p_1$ **except** $p_2$ denotes the difference of $p_1$ and $p_2$.
4. $p_1.c[.i]$ projects $p_1$ on the $i^{th}$ coordinate of component $c$. When there is only one component, its name can be omitted.
5. $p_1.c.i$ **as** $j$ renames the $i^{th}$ coordinate of $c$ as $j$.

**join** is the most general operation: depending on the existence of common components in $p_1$ and $p_2$, one obtains a cartesian product (no common components), a join (some common components), or an intersection (same components). In the latter case the equivalent **inter** keyword can be used for the sake of clarity.

The renaming of components or variables is obtained with **as** and allows to control the semantics of the **join**. Observe that the (spatial) selection is obtained as ($p_1$ **join** *cst*) where *cst* is any formula that represents a point set. For clarity, we sometimes use the equivalent syntax **restrict** $p_1$ **with** *cst*.

We can use the boolean counterpart of an operator **op**, denoted **op?**, in the **where** clause. It returns **true** if the resulting set is non empty and **false** otherwise. Therefore the general form of a query is

**select** $att_1, \ldots, att_n, E_1, \ldots E_m$
**from** $R_1[, ...]$
**where** $B_1, \ldots B_l$

where $att_i$ denotes atomic attributes, $E_j$ algebraic expressions over nested attributes, and $B_k$ boolean algebraic expressions.

### 5.5.2 Example Queries

Let us now illustrate the use of our nested relational model and SQL query language in the real-life application of Example 1 [8]. The data modeling of this application is quite simple. It consists of two collections of objects:

1. A classical map (2-dimensional spatial objects) that describes the buildings and areas of interest in the ski resort.
2. A set of moving objects, each of which represents some typical socio-economical behavior. For simplicity, we will consider only two such objects named John and Monica.

Note that moving objects contain two time-varying attributes: their activity (pure relational information) and their geometry (shape and position). Observe also that while the shape here is irrelevant since we represent each object by a single point, nothing in the model prevents us from describing complex shaped moving objects. We give below the DEDALE schema for this application:

**Relation** Resort
(name: **string**,
geom: (space: **float(2)**)
)

**Relation** People
(name: **string**
activity: (alpha: **string**,
time: **float(1)**)
traj : (space: **float(2)**,
time: **float(1)**)
)

Although extremely simple, this schema allows for a wide range of queries illustrating various combinations of spatial, temporal and relational criteria. We give a sample list of these queries in the sequel, along with some comments on either query design or its underlying evaluation in DEDALE. The same application gives rise to complex temporal relationships and queries which have been considered in the context of the TEMPOS project (see [13,41]).

All queries below run in the current implementation of DEDALE.

1. *Where is John between 10 and 12?*

   **select** (**restrict** traj **with** '10 < time < 12').space
   **from** People
   **where** name = 'John'

   A temporal query with spatial output: the constraint '10 < time < 12' is added to the trajectory of John, and tuples in the resulting *traj* relation which are still satisfiable (if any) are projected on the *space* component.
2. *When does Monica stay at the bar's terrace?*

   **select** (p.traj **join** r.geom).time
   **from** Resort r, People p
   **where** p.name = 'Monica'
   **and** r.name = 'Terrace'

A spatial query with temporal output. The spatial join on the *space* component common to *t.traj* and *s.geom* yields the part of Monica's trajectory inside the terrace's geometry: its projection on *time* is the result.

3. *Where is John while Monica is at the bar's terrace?*

```
select  (p2.traj  join (p1.traj join r.geom).time).space
from    Resort r, People p1, People p2
where   p1.name = 'Monica'
and     p2.name = 'John'
and     r.name = 'Terrace'
```

This query is a composition of a spatial join and a temporal join. The internal join retrieves time $t$ during which Monica is at the terrace; $t$ is the input argument to the temporal join with John's trajectory.

4. *Show the places where Monica sleeps*

```
select ((restrict activity with "alpha = 'Sleeping' ") join traj).space
from   People
where  name = 'Monica'
```

Here, a pure temporal query based on alphanumerical criteria ("retrieve the periods that correspond to a given activity") is composed with a temporal join. Note that the temporal join is "internal" to a single object since it involves the two nested relations (*activity* and *traj*) of Monica.

5. *Where did Monica and John meet?*

```
select  (p1.traj inter p2.traj).space
from    People p1, People p2
where   p1.name = 'Monica'
and     p2.name = 'John'
```

The join involves simultaneously time and space.

6. *When were Monica and John at the same place?*

```
select  (p1.traj join r.geom).time inter (p2.traj join r.geom).time
from    Resort r, People p1, People p2
where   p1.name = 'Monica'
and     p2.name = 'John'
```

Each object in the ski resort map is intersected with the trajectories of John and Monica. This yields the places where both of them spent some time during a typical day. A further join on *time* restricts the result to the places where they both were at the same instant.

7. *Who ate in the skiing area, and when?*

```
select  ((restrict p.activity with "alpha = 'Eating' " join p.traj)
          join r.geom).time, p.name
from    Resort r, People p
where   r.name = 'Skiing Area'
```

A first internal temporal join between the two time-varying attributes of a moving actor gives places where this actor eats. The following spatial join checks whether these places intersect the skiing area.

8. *What did John between the ski and his dinner?*

**selectect** name,
(((activity **join** "alpha = 'Ski' ").[time **as** t1]
**join** activity
**join** (activity **join** "alpha = 'Eat' ").[time **as** t2]
) **join** "t1.1 $\leqslant$ time.1 $\leqslant$ t2.1").alpha
**from** People
**where** name = 'John'

The comparison between three time components entails both a blow-up in dimension due to the cross-products and an unrestricted selection that links variables coming from different components. This yields an intermediate result whose global dimension is 9. Fortunately, the evaluation techniques over dimension-restricted queries developed in [17] allow to apply only operations on time intervals. In that case, the evaluation is as follows: $t1.1$ should be less than $Max(time.1)$ and $t2.1$ should be greater than $Min(time.1)$. The result is correct, as long as a projection of some component (in that case *alpha*) is made as a last operation, as required for dimension-restricted queries. The description of the evaluation techniques for such queries is beyond the scope of this chapter (the interested reader is referred to [17]).

## 5.6 Conclusions

This chapter presented the constraint-based approach to representing and querying spatio-temporal data. Three different models were discussed: the original constraint database model of [25], the indefinite constraint database model of [28,27] and the DEDALE model [16]. In all cases we used linear (and in one case polynomial) constraints to represent spatio-temporal data in a uniform way. This can be contrasted with the approach of Chapter 4 where an abstract data type is devised for each useful spatio-temporal concept.

Similar views to the ones developed in this chapter have also been expressed by other constraint-database researchers. Data models and query languages similar to the ones developed for DEDALE have also been presented in [3,4] and [34]. DEDALE is currently the only implemented system based on these ideas.

Spatio-temporal data have also been studied in [9,11,7] where a spatio-temporal model based on parametric rectangles is proposed. The same group of researchers has developed the constraint database system MLPQ-GIS [38] which is able to deal with spatio-temporal applications. The main difference of this system from DEDALE is that it is based on flat relations and the original constraint database model and uses a query language based on DATALOG [38]. In a related paper the issue of interoperability of spatio-temporal data is discussed [10]. This is an interesting topic not covered in this chapter.

We should also mention the constraint database system CCUBE which is based on an object-oriented model and also uses linear constraints for the representation of spatio-temporal objects [5,6].

## References


1. M. Baudinet, J. Chomicki, and P. Wolper. Temporal Deductive Databases. In Tansel, A., et al., editor, *Temporal Databases: Theory, Design, and Implementation*, chapter 13. Benjamin/Cummings Pub. Co., 1993.
2. C. Beeri. Data Models and Languages for Databases. In *Proceedings of 2nd International Conference on Database Theory (LNCS 326)*, pages 58–67, 1989.
3. A. Belussi, E. Bertino, and B. Catania. Manipulating Spatial Data in Constraint Databases. In M. Scholl and A. Voisard, editors, *Proc. of the Fifth Int. Symp. on Spatial Databases*, number 1262 in Lecture Notes in Computer Science, pages 115–140, Berlin, Germany, July 1997. Springer Verlag, Berlin.
4. A. Belussi, E. Bertino, and B. Catania. An Extended Algebra for Constraint Databases. *IEEE Transactions on Knowledge and Data Engineering*, 10(5):686–705, 1998.
5. A. Brodsky and V.E. Segal. The CCUBE Constraint Object-Oriented Database System: an Overview. *Constraints*, 2(3-4):245–277, 1997.
6. A. Brodsky, V.E. Segal, J. Chen, and P.A Exarkhopoulo. The CCUBE Constraint Object-Oriented Database System. In *Proceedings of SIGMOD 1999*, pages 577–579, 2000.
7. M. Cai, D. Keshwani, and P.Z. Revesz. Parametric rectangles: A model for querying and animation of spatio-temporal databases. In *Proceedings of EDBT 2000*, pages 430–444, 2000.
8. S. Chardonnel and P. Dumolard. Personal communication.
9. J. Chomicki, Y. Liu, and P.Z. Revesz. Animating spatiotemporal constraint databases. In *Spatio-Temporal Database Management (Proceedings of the International Workshop STDBM'99)*, volume 1678 of *LNCS*, pages 224–241. Springer, 1999.
10. J. Chomicki and P.Z. Revesz. Constraint-based Interoperability of Spatiotemporal Databases. *GeoInformatica*, 3(3):211–243, 1999.
11. J. Chomicki and P.Z. Revesz. A geometric framework for specifying spatiotemporal objects. In *Proceedings of TIME'99*, pages 41–46, 1999.
12. R. Dechter, I. Meiri, and J. Pearl. Temporal Constraint Networks. *Artificial Intelligence*, 49(1-3):61–95, 1991. Special Volume on Knowledge Representation.
13. M. Dumas, M.-C. Fauvet, and P.-C. Scholl. Handling Temporal Grouping and Pattern-Matching Queries in a Temporal Object Model. In *Proc. Intl. Conf. on Information and Knowledge Management*, pages 424–431, 1998.
14. G. Grahne. The Problem of Incomplete Information in Relational Databases. Technical Report Report A-1989-1, Department of Computer Science, University of Helsinki, Finland, 1989. Also published as Lecture Notes in Computer Science 554, Springer Verlag, 1991.
15. S. Grumbach, P. Rigaux, M. Scholl, and L. Segoufin. DEDALE: A Spatial Constraint Database. In *Proc. Intl. Workshop on Database Programming Languages*, pages 38–59, 1997.
16. S. Grumbach, P. Rigaux, and L. Segoufin. The DEDALE System for Complex Spatial Queries. In *Proc. ACM SIGMOD Symp. on the Management of Data*, pages 213–224, 1998.

17. S. Grumbach, P. Rigaux, and L. Segoufin. On the Orthographic Dimension of Constraint Databases. In *Proc. Intl. Conf. on Database Theory*, pages 199–216, 1999.
18. S. Grumbach, J. Su, and C. Tollu. Linear Constraint Query Languages: Expressive Power and Complexity. In D. Leivant, editor, *Logic and Computational Complexity*, Indianapolis, 1994. Springer Verlag. LNCS 960.
19. R.H. Güting. Gral: An Extensible Relational Database System for Geometric Applications. In *Proc. Intl. Conf. on Very Large Data Bases (VLDB)*, 1989.
20. R.H. Güting and M. Schneider. Realm-Based Spatial Data Types: The ROSE Algebra. *The VLDB Journal*, 4(3):243–286, 1995.
21. J. Herring. The ORACLE 7 Spatial Data Option. Technical report, ORACLE Corp., 1996.
22. G.E. Hughes and M.J. Cresswell. *An Introduction to Modal Logic*. Methuen, 1968.
23. T. Imielinski and W. Lipski. Incomplete Information in Relational Databases. *Journal of ACM*, 31(4):761–791, 1984.
24. F. Kabanza, J.-M. Stevenne, and P. Wolper. Handling Infinite Temporal Data. In *Proceedings of ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 392–403, 1990. Full version appears in JCSS, Vol. 51, No. 1., pages 3-17, 1995.
25. P.C. Kanellakis, G.M. Kuper, and P.Z. Revesz. Constraint query languages. In *Proc. ACM Symp. on Principles of Database Systems*, pages 299–313, 1990. A longer version appears in JCSS, Vol. 51, No. 1, 1995.
26. M. Koubarakis. Representation and Querying in Temporal Databases: the Power of Temporal Constraints. In *Proceedings of the 9th International Conference on Data Engineering*, pages 327–334, April 1993.
27. M. Koubarakis. Database Models for Infinite and Indefinite Temporal Information. *Information Systems*, 19(2):141–173, March 1994.
28. M. Koubarakis. Foundations of Indefinite Constraint Databases. In A. Borning, editor, *Proceedings of the 2nd International Workshop on the Principles and Practice of Constraint Programming (PPCP'94)*, volume 874 of *Lecture Notes in Computer Science*, pages 266–280. Springer Verlag, 1994.
29. M. Koubarakis. Databases and Temporal Constraints: Semantics and Complexity. In J. Clifford and A. Tuzhilin, editors, *Recent Advances in Temporal Databases (Proceedings of the International Workshop on Temporal Databases, Zürich, Switzerland, September 1995)*, Workshops in Computing, pages 93–109. Springer, 1995.
30. M. Koubarakis. The Complexity of Query Evaluation in Indefinite Temporal Constraint Databases. *Theoretical Computer Science*, 171:25–60, January 1997. Special Issue on Uncertainty in Databases and Deductive Systems, Editor: L.V.S. Lakshmanan.
31. M. Koubarakis and S. Skiadopoulos. Querying Temporal Constraint Networks in PTIME. In *Proceedings of AAAI-99*, pages 745–750, 1999.
32. M. Koubarakis and S. Skiadopoulos. Tractable Query Answering in Indefinite Constraint Databases: Basic Results and Applications to Querying Spatio-Temporal Information. In *Spatio-Temporal Database Management (Proceedings of the International Workshop STDBM'99)*, volume 1678 of *LNCS*, pages 204–223. Springer, 1999.
33. G. Kuper, L. Libkin, and J. Paredaens, editors. *Constraint Databases*. Springer-Verlag, 2000.
34. G. Kuper, S. Ramaswamy, K. Shim, and J. Su. A Constraint-Based Spatial Extension to SQL. In *Proc. Intl. Symp. on Geographic Information Systems*, 1998.

35. J. Orenstein and F. Manola. PROBE: Spatial Data Modeling and Query Processing in an Image Database Application. *IEEE Transactions on Software Engineering*, 14(5):611–628, 1988.
36. J. Paredaens, J. Van den Bussche, and D. Van Gucht. Towards a Theory of Spatial Database Queries. In *Proc. 13th ACM Symp. on Principles of Database Systems*, pages 279–288, 1994.
37. P.Z. Revesz. A Closed Form Evaluation for Datalog Queries with Integer (Gap)-Order Constraints. *Theoretical Computer Science*, 116(1):117–149, 1993.
38. P.Z. Revesz, R. Chen, P. Kanjamala, Y. Li, Y. Liu, and Y. Wang. The MLPQ/GIS Constraint Database. In *Proceedings of SIGMOD 2000*, 2000.
39. N. Roussopoulos, C. Faloutsos, and T. Sellis. An Efficient Pictorial Database System for PSQL. *IEEE Transactions on Software Engineering*, 14(5):639–650, 1988.
40. M. Scholl and A. Voisard. Thematic Map Modeling. In *Proc. Intl. Symp. on Large Spatial Databases (SSD)*, LNCS No. 409, pages 167–192. Springer-Verlag, 1989.
41. P.-C. Scholl, M.-C. Fauvet, and J.-F. Canavaggio. Un Modèle d'Historique pour un SGBD Temporel. *TSI*, 17(3), mars 1998.
42. A. Schrijver. *Theory of Linear and Integer Programming*. Wiley, 1986.
43. A.P. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Modeling and Querying Moving Objects. In *Proceedings of ICDE-97*, 1997.
44. R.T. Snodgrass, editor. *The TSQL2 Temporal Query Language*. Kluwer Academic Publishers, 1995.
45. A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. Snodgrass, editors. *Temporal Databases: Theory, Design, and Implementation*. Database Systems and Applications Series. Benjamin/Cummings Pub. Co., 1993.
46. D. Toman, J. Chomicki, and D.S. Rogers. Datalog with Integer Periodicity Constraints. In *Proceedings of the International Symposium on Logic Programming*, pages 189–203, 1994.
47. M. Ubell. The Montage Extensible DataBlade Architecture. In *Proc. ACM SIGMOD Intl. Conference on Management of Data*, 1994.