# Thematic Map Modeling[1]

Michel Scholl and Agnès Voisard
I.N.R.I.A., 78153 Le Chesnay, France
(scholl, voisard)@bdblues.altair.fr

**Abstract**

We study here how to provide the designer of geographic databases with a database query language extensible and customizable. The model presented here is a first step toward a high level spatial query language adapted to the manipulation of thematic maps.

For this, we take as an example a toy application on thematic maps, and show by using a complex objects algebra that application dependent geometric operations can be expressed through an extension of the *replace* operator of [AB88].

## 1 Introduction

The representation and manipulation of geometric information require the use of two technologies: database systems and computational geometry. Several recent proposals have been made for the modeling and design of Geographic Information Systems (GIS) (see for example [LM84], [OM86], [SW86], [MOD87], [Dav88], [RFS88]). For a comprehensive study on the requirements for the design and implementation of large-scale GIS, see [Fra84], [SMSE87]. Several spatial query languages have recently been proposed in [CF80], [CK81], [BB81], [Fra82], [SDMO87], [CJ88], [Gut88]. A survey on data structures for spatial databases can be found in [Sam84].

One characteristics of GIS is that they cover an extremely wide range of applications for which, neither a common definition of objects, nor a common set of functions on these objects exist. Designing a close general information system for geographic applications therefore becomes an ambitious and somewhat hazardous task.

It is our belief that there might exist a set of *application dependent* basic objects and operations on these objects, and that the database system should permit an easy extension of this set or an easy change to another set more adapted to a particular application. Examples

[1]This work was partially supported by a grant from the french PRC $BD_3$, and by the BRA ESPRIT W.G. Basic GOODS.

of applications we consider are cartography and urban planning which both manipulate thematic maps.

In order to validate the concepts presented in the paper, we consider a restricted application with limited functionalities since it manipulates only regions, i.e. subsets of $R^2$ (and neither lines nor points).

This application is described in Section 2. Basically, one would like to answer queries such as:

- "*Display the districts of the province of Toulouse*",
- "*From the map of the districts of France, zoom to the map of provinces*",
- "*Overlay the map of crops with the map of the province of Rennes*",
- *"Create the map of districts from the map of provinces belonging to the North of France" (district boundaries as well as data associated to districts must be entered)*,
- *"Display districts with more than* 20% *of the people voting for the communist party in the district of Lille"*.

For designing such an application, the following are required:

- some high level query and manipulation language with the confidence that, when the application changes (and therefore the query language changes), there are only minimal incremental changes to bring to the system in order to provide new functionalities,
- a powerful manipulation of a large amount of data.

A candidate approach is to take an "extended" relational system where we model geometric information by means of specific attributes with the two-dimension space for domain. By *extended* we mean that the database system is augmented with capabilities such as defining and running specific functions associated with a given domain (see for example [SR86, GCK+89]). Then by attaching application dependent functions to geometric objects, one may hope to design specific GIS.

Augmenting the relational query language by specific geometric operators was, to our knowledge, first proposed in [CF80]. In [SRG83], abstract data types are proposed for geometric objects and their operations. A similar approach to [SRG83] is described in [Ore86]. But the most significant integration of geometric data type and operators into a relational algebra is due to [Gut88].

However such approaches suffer from the following drawbacks:

1. Data structures represented by relational systems are very poor. More recent database approaches such as complex object models [HY84], [KV85], [BK86], [Hul87], [AG87],

[AB88]), or object oriented systems (see for example [K+88], [K+89], [BBB+88], [LRV88], [LR89]) are more powerful for representing complex data structures.

2. User defined operations are not part of the relational model. This is why extended relational systems are currently designed. The disadvantage of such an approach is its lack of flexibility: once the extended relational system has been designed, it becomes cumbersome to adapt it to new functionalities required by other applications and users.

Although object oriented systems provide the representation of complex structures and a flexible definition of user defined operations, they do not yet provide query languages. Their interface can still be considered as a low level approach not adapted to unsophisticated users who require a high level language for spatial queries.

The purpose of this paper is to show that user defined geometric operations can be embedded into a very general database query language, in a simple and flexible way.

In some sense, our model can be seen as continuing the work reported in [Gut88]: we believe that an algebraic approach is a powerful tool for efficiently manipulating large sets of data. However, our approach differs from the above work in two aspects:

1. To relational algebra, we prefer a complex object algebra because of its expressive power.

2. Such a complex object model is still unsufficient for representing user defined geometric operations. This is why we suggest **not** to embed the geometric operations into the data model, but rather to associate them to the data model through simple and general constructs.

As far as implementation of the spatial query language is concerned, object-oriented systems are good candidates. But object-oriented features such as object identity or inheritance are not necessary at the language level.

The paper is organized as follows: Section 2 describes the application chosen for illustrating our approach. The model presented in Sections 3 and 4 is a first step towards the definition of a high level spatial query language adapted to the manipulation of thematic maps. Recall that the objective is to associate a powerful database modeling tool independent of the (geometric) application, to an adaptative manipulation of geometric objects. It is our belief that the same methodology can be applied for the design of GIS manipulating lines.

In Section 3, we define what is a map and which are the operations on maps, i.e. we define a query language on maps. We also define regions as well as a set of operations on regions. We show that these operations on maps are sufficient to implement primitives such as those described in Section 2.

In Section 4, we show that these operations on maps apparently closely related to the application can be expressed by means of very general constructs: the approach followed for modeling maps is basically a *complex objects* approach. Among all existing models we chose the model of [AB88]. Such a model is adequate for representing and manipulating databases whose objects have various complex structures. Besides it includes a powerful operator called *replace*. This operator applies a function to each element of a set of objects. This operation is necessary for expressing elaborate operations on complex structures. But classical operations such as selection and projection can be expressed by a *replace* as well.

The function specification in *replace* is **not** user defined in [AB88]. We extend this specification to express various user defined (geometric) functions, and call it *apply*. This provides independence between (i) an algebraic language for general (non geometric) data manipulations on maps independently of the application and (ii) the operations on basic geometric objects specific to a given application.

Section 4 defines the *apply* construct and shows how the operations on maps described in Section 2 are expressed through the *apply* construct. The map representation chosen may imply certain redundancies. They are discussed in Section 5.

## 2 A toy application

In this Section, we informally describe a few primitives for thematic maps manipulation. A more formal presentation will be given in Section 3. We just assume here that a map has two kinds of components: non-spatial and spatial components. The former take alphanumerical values, while the latter represent geometric regions.

The operations described here are the following: projection, fusion, cover, map overlay, superimposition, selection, windowing, and finally clipping.

- **Projection:**

This operation corresponds to the relational projection. As an example, take map $m$ of figure 2.1. The attributes "crop" and "district" whose domains are respectively "string" and "integer" represent its non-spatial components. Assume that we want to get the map of districts, without mention of the crops of each district: we have to project out attribute "crop".

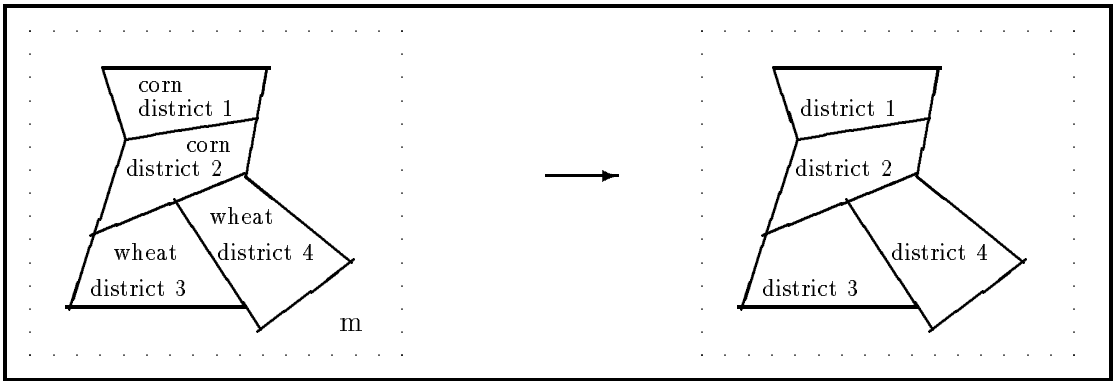



Figure 2.1: projection on attribute "district"

Assume now that instead of projecting the previous map on districts, we want to get the map of crops (without mention of the districts). After having applied the projection operation, two neighbour regions may have the same crop value. Then we may want to replace such neighbour regions by a single region, i.e. erase their common boundary. The **fusion** operation realizes the geometric union of the regions of a map which have the same value for a given component. Figure 2.2.a represents the map m of crops and districts. Figure 2.2.b represents the map of crops after projecting out the district name and realizing the fusion of regions with common crop value: we do not distinguish anymore districts boundaries.

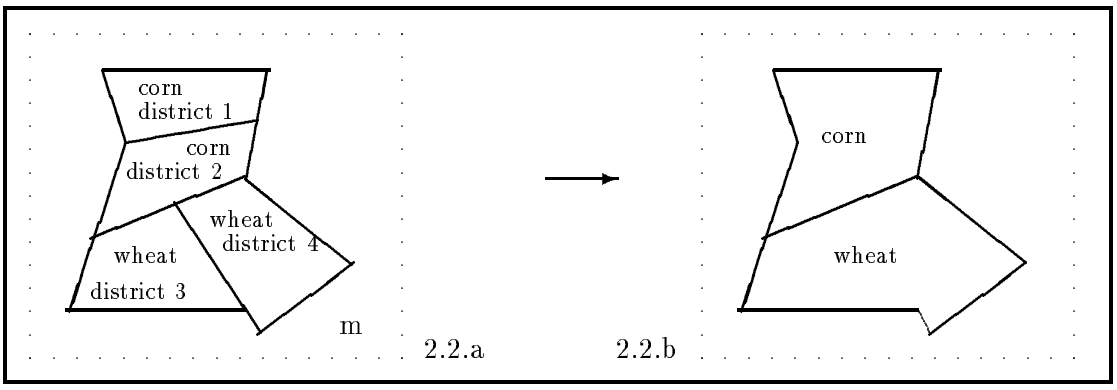



Figure 2.2: projection and fusion of map $m$ on attribute "crop"

If we further fusion the two regions of Figure 2.2.b, after projecting out attribute "crop", we obtain the **cover** of m, i.e. the (geometric) union of the regions of $m$ (Fig 2.3).

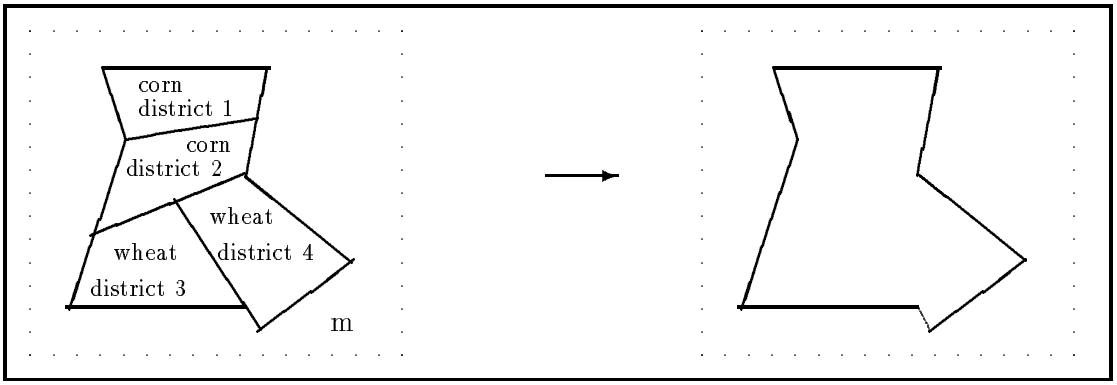



Figure 2.3: cover of map m

- **Map overlay:**

Figure 2.4 exhibits an example of map overlay on maps with same cover. In the case where both maps do not have the same cover, the cover of the resulting map would be the intersection of the covers of the two maps.

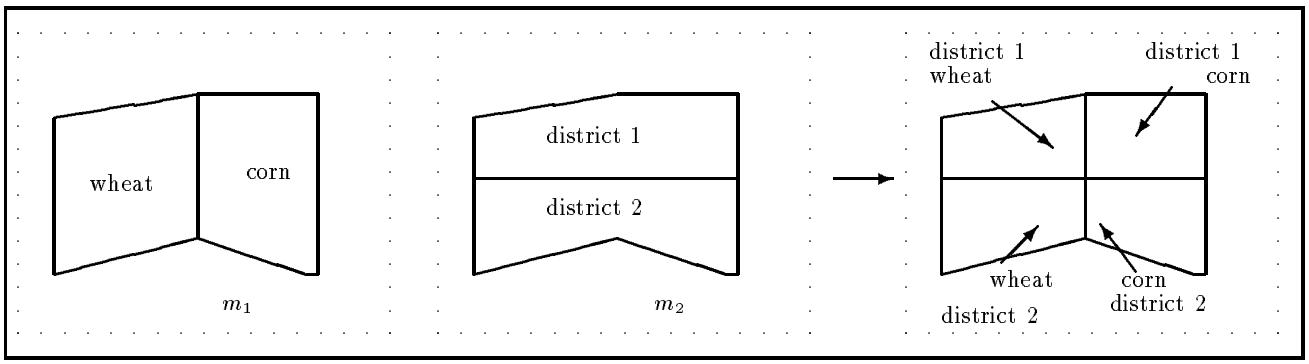



Figure 2.4: map overlay between $m_1$ and $m_2$

- **Superimposition:**

Superimposing a text, an icon, a caption or a map onto a given map is useful in cartography. Figure 2.5 shows an example of superimposition of a map $m_2$ onto a map $m_1$. We assume that the cover of $m_2$ is included in the cover of $m_1$.

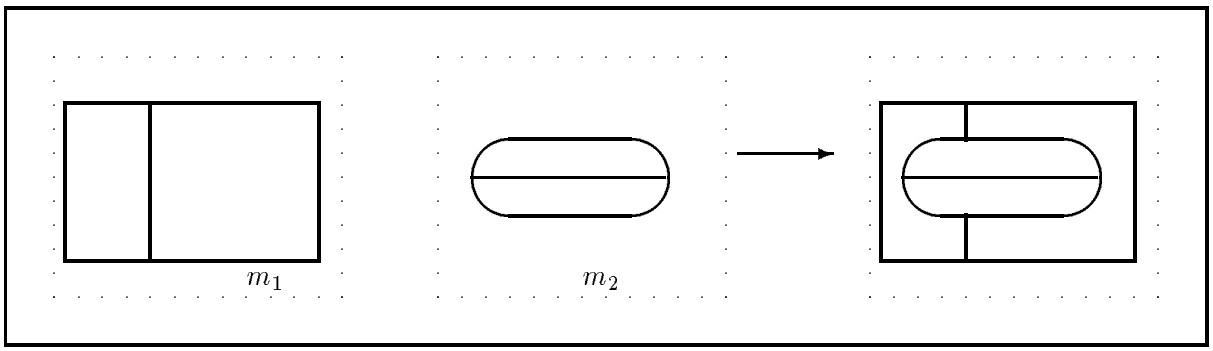



Figure 2.5: superimposition of $m_2$ onto $m_1$

- **Selection:**

Consider again the example of Figure 2.1. Getting the map of districts producing "wheat" corresponds to the relational selection (Figure 2.6).

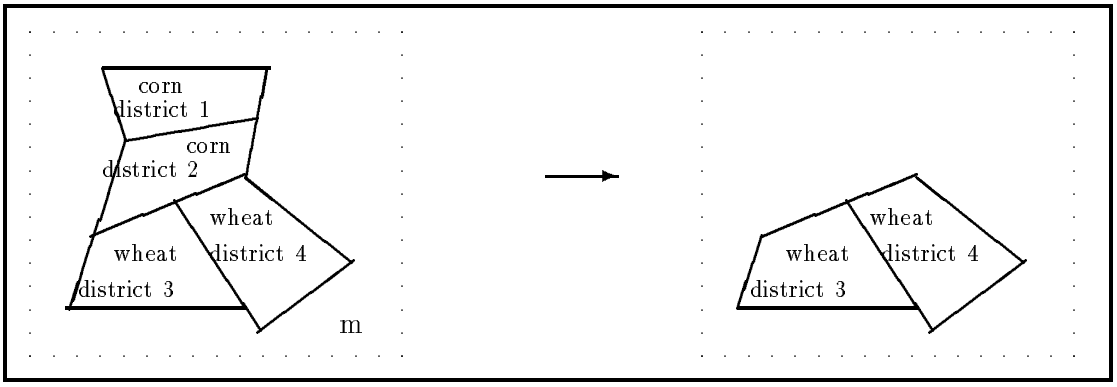



Figure 2.6: example of relational selection

The following operations are examples of *geometric* selections:

- **Windowing:**

Windowing (Figure 2.7) allows to get the regions of a map whose intersection with a given window is not empty.

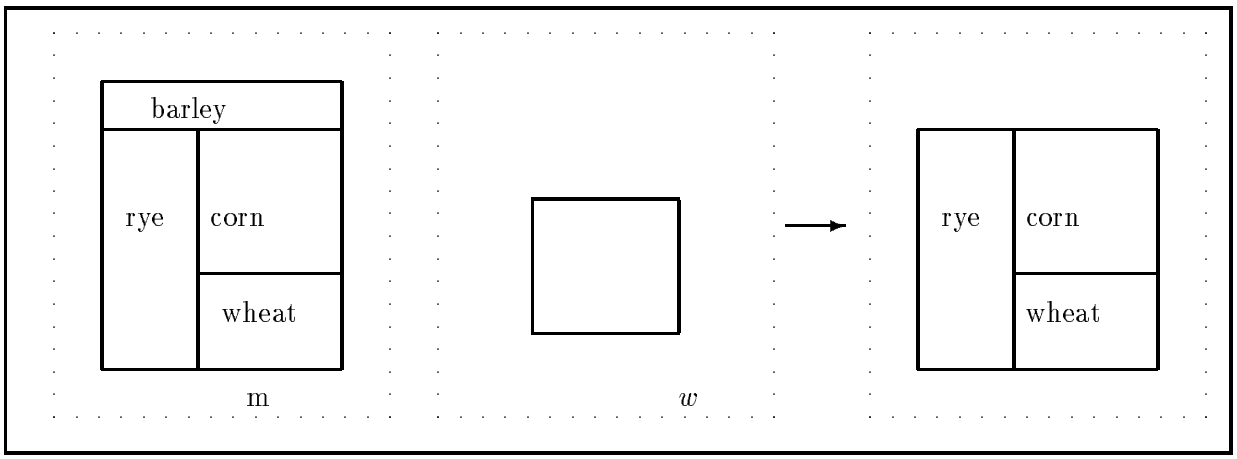



Figure 2.7: windowing of map $m$ with window $w$

As another example, consider the query "what are the communes whose distance to a given point $p$ is less than $r$=twenty kilometers". Windowing is applied where the window is the circle with center $p$ and radius $r$:

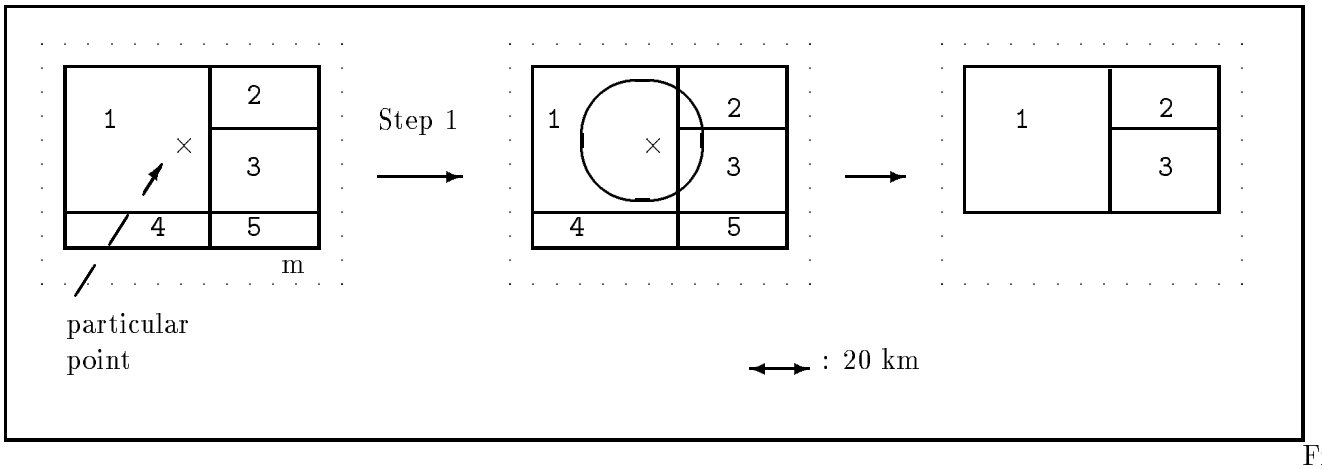



Figure 2.8: "communes whose distance to a given point is less than 20 kilometers"

- **Clipping:**

This operation allows to select the part of a map which is inside a window.

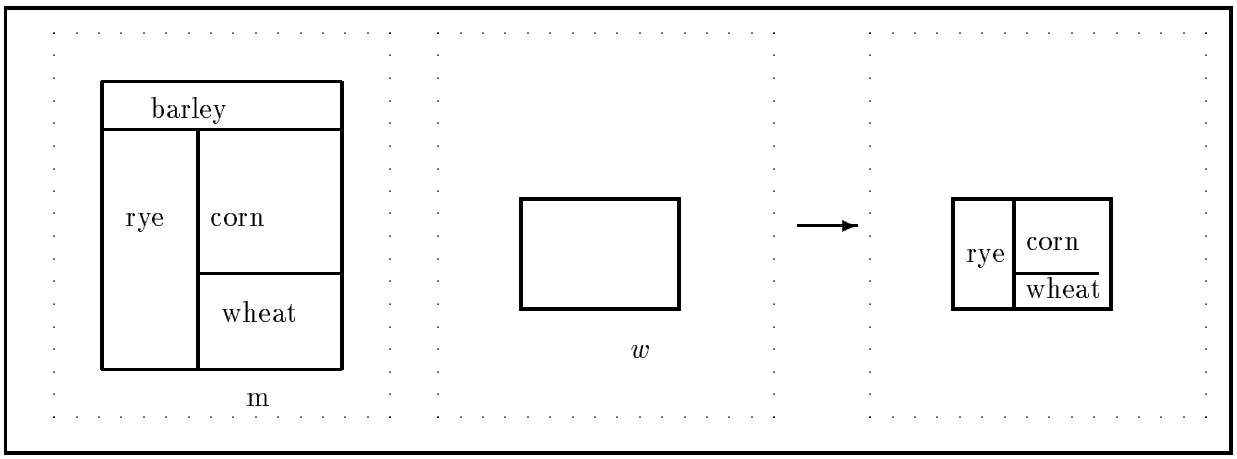



Figure 2.9: clipping of a map $m$ with a window $w$

# 3 The model

We are interested in querying a set of thematic maps in two distinct ways: (i) issuing queries with respect to the alphanumeric information associated to maps (e.g. relational selection), and (ii) asking information related to the geometric component of the map (e.g. geometric selection, map overlay). Database systems efficiently manipulate sets of objects. This is why it sounds natural to represent a thematic maps database by a set of *maps*, where a map is a set of *tuples*. A tuple basically represents a geometric region to which is usually associated non geometric information.

In this section, we introduce a data model on maps. Operations on maps are database operations extended with geometric operations on regions.

We first introduce regions and operations on regions (Section 3.1). The data model on maps is then defined (Section 3.2). Finally, Section 3.3 illustrates this map data model on the application described in Section 2.

**Notations**
In the sequel, we adopt the following conventions: we use greak letters, $\gamma$ for a geometric type, and $\tau$, $\mu$,... for a non geometric type.

Capital letters such as A, B, ... are used for alphanumeric attribute names, and R, S, ... for geometric ones. Values are denoted by lower case letters.

## 3.1 Regions

### 3.1.1 Definition of a region

An *elementary region* is a subset of the two-dimension space $R^2$ (figure 3.1). It can be :

- a polygon (e.g. $r_3$, $r_4$),

- a subset of $R_2$ bounded by lines (e.g. $r_1$, $r_2$),
- the union over $R^2$ of several connected or non connected parts of the plane (e.g. $r_3 \cup r_4$).

A *region* can be:

- an elementary region, or
- a set of elementary regions.

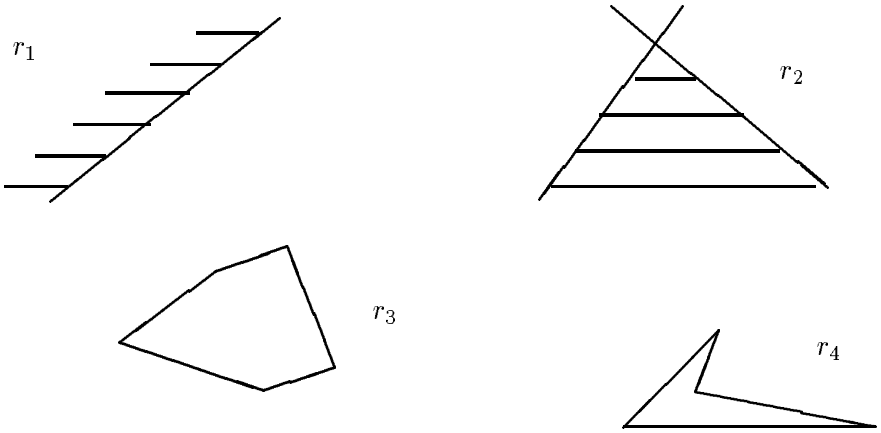



Figure 3.1: a few examples of elementary regions

Let us consider now the following example:

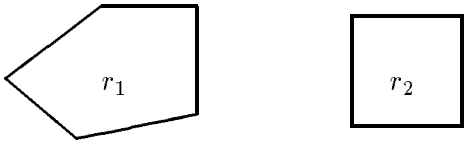



Figure 3.2: elementary regions or set of regions

Looking at Figure 3.2, we may choose between the three following interpretations: (i) $r_1$ and $r_2$ are two distinct elementary regions, (ii) the geometric union of $r_1$ and $r_2$ is an elementary region, and (iii) the set of elementary regions $\{r_1, r_2\}$ is a region.

More precisely, let $\gamma$ be a given subset of $R^2$.

- $\gamma$ is a region type, called atomic type: an elementary region is a region of type (domain) $\gamma$.
- $\{\gamma\}$ is a region type: a set of elementary regions is a region of type $\{\gamma\}$.

### 3.1.2 Operations on regions

We introduce now a set of operations on regions which is sufficient to implement the application of Section 2, and which will be useful to describe operations on maps:

1. we first use a boolean algebra over $R^2$, where the operators +, • and - are interpreted by the corresponding set operations: union, intersection and difference:

   (a) + (union) :
   $r_1$, $r_2$, $r_1 + r_2$ of type $\gamma$.

   (b) • (intersection):
   $r_1$, $r_2$, $r_1 \bullet r_2$ of type $\gamma$.

   (c) - (difference):
   $r_1$, $r_2$, $r_1 - r_2$ of type $\gamma$.

   We then extend this algebra with the following operations:

2. we introduce the "non-empty" unary predicate over $\gamma$, denoted $\neq \emptyset$.
   $r \neq \emptyset$ is true if $r$ is not the empty subset of $R^2$, where r is of type $\gamma$.

   Similarly, if $r$ is of type $\{\gamma\}$, $r = \{s_1, ...s_n\} \neq \{\emptyset\}$ if there is at least one $s_i$ such that $s_i \neq \emptyset$.

3. $\oplus : \{\gamma\} \rightarrow \gamma$.
   $\oplus\{r_1, ..., r_n\} \stackrel{\text{def}}{=} r_1 + ... + r_n$.
   This operator performs the union over $R_2$ of the elementary regions elements of a set and gives as an output a single elementary region.

4. $\sqcap$ (set from singleton): $\gamma \rightarrow \{\gamma\}$.
   $\sqcap r \stackrel{\text{def}}{=} \{r\}$.
   This operator transforms an elementary region r into a set with unique region r.

5. intersection is extended by:

   (a) $\{\gamma\} \times \gamma \rightarrow \{\gamma\} : \{r_1, ..., r_n\} \bullet r \stackrel{\text{def}}{=} \{r_1 \bullet r, ..., r_n \bullet r\}$.

   (b) $\gamma \times \{\gamma\} \rightarrow \{\gamma\} : r \bullet \{r_1, ..., r_n\} \stackrel{\text{def}}{=} \{r \bullet r_1, ..., r \bullet r_n\}$.

   (c) $\{\gamma\} \times \{\gamma\} \rightarrow \{\gamma\} : r \bullet s \stackrel{\text{def}}{=} \{t \bullet u \mid t \in r,\ u \in s\}$, where $r$, $s$ are of type $\{\gamma\}$ and $t$, $u$ are of type $\gamma$.

## 3.2 Maps

### 3.2.1 Definition

A map is a set of tuples. As an example of tuple, $[A : a,\ B : b,\ R : r]$ will designate a region with geometric attribute R with value $r$ and with two non geometric attributes $A$ and $B$ with respective values $a$ and $b$.

We do not require the existence of non geometric attributes (e.g. $\{[R : r]\}$ represents a map including a single region). However a map must have at least one geometric attribute.

We accept maps with several geometric attributes. This is explained in Section 5.

To define more formally a map, we follow the *complex objects* approach of [AB88]. Maps are typed objects defined as follows.

**Types and objects**
We assume an infinite set of attribute names (geometric or not), a given set of non-geometric domains $\{D_1, ..., D_n\}$ and a given set of geometric domains $\{\triangle_1, ..., \triangle_m\}$, where $\triangle_i \subseteq R^2$.

Types are constructed from domains, attribute names, and the set {} and tuple [ ] constructors. Each object is an instance of a type which is defined as follows:

1. if D ($\triangle$) is a domain, then D ($\triangle$) is a type (geometric or not).
2. if $\tau_1$, ..., $\tau_n$ are types and $L_1$, ..., $L_n$ are attributes names not used in any of them, then $[L_1 : \tau_1, ..., L_n : \tau_n]$ is a tuple type.
3. if $\tau$ is a type and L an attribute name not used in it, then $\{L : \tau\}$ is a set type.
4. if $\tau$ is a type and L a name not used in it, then L : $\tau$ is a named type.

We illustrate this definition by examples of maps and associated types:
$\{R : r, R : s\}$ of type $\{R : \gamma\}$
{[A : a, B : b, R : r, S : s]} of type $\{[A : string, B : string, R : \gamma, S : \gamma]\}$
$S : \{[A : a, B : \{C : 1, C : 2\}, R : r]\}$ of type
S : $\{[A : string, B : \{C : integer\}, R : \gamma]\}$

### 3.2.2 Operations on maps

To define operations on maps, the algebra of [AB88] is extended with "geometric" operations. Let us consider first the standard database operations. Among the useful operations are the following ones: projection, selection, cartesian product, nest and set operations. We recall the usual notations for the four first operations:

1. **Relational algebra operations**:
   (a) *Projection* is denoted by $\pi_{list\ of\ attribute\ names}$. The value of the component $A$ of tuple $t$ of map $m$ is denoted: $t.A$.
   (b) *Selection* is denoted by $\sigma_{condition}$. For example, "all the regions of a map $m$ whose altitude is more than 1000 meters" where $m$ is of type: $\{[Height : integer, Crop : string, R : \gamma]\}$ is expressed by:
   $$"\pi_R(\sigma_{Height>1000}(m))".$$
   (c) *Cartesian product* is denoted by $\times$. For example, $m_1 \times m_2$ where $m_1$ is of type[2] $\{[A, R_1]\}$ and $m_2$ is of type $\{[B, R_2]\}$ will generate a map $m$ of type $\{[A, B, R_1, R_2]\}$.

[2]From now on, when there is no ambiguity, we omit in a tuple type, the type of each component; for example, $\{[A, R]\}$ denotes type $\{[A : String,\ R : \gamma]\}$.

2. **Nesting** map $m$ on attribute $A$ is denoted $Nest_A(m)$. As an example, if $m$ is of type $\{[A : String,\ R : \gamma]\}$ nesting $m$ on attribute $A$ will group in a single tuple all regions having the same value for attribute $A$. $Nest_A(m)$ is of type $\{[A : String, S : \{R : \gamma\}]\}$.

"Geometric" operations on maps are now introduced:

1. **Fusion** $\uplus$:
   without loss of generality, consider a map $m$ of type $\{[A : \tau, R : \{\gamma\}]\}$. The fusion operation replaces in each tuple of $m$ the set of regions by its geometric union $\oplus$. Let $m' = \uplus(m)$. $m'$ is of type $\{[A : \tau, R' : \gamma]\}$ and has the following value:

$$m' = \{t' | t \in m, t'.A = t.A \wedge t'.R' = \oplus t.R\}$$

2. **Geometric selection** denoted by $\sigma_G$, where $G$ stands for geometric condition.
   $G$ is one of the following predicates[3]:
   $$t.R \bullet k \neq \emptyset,$$
   $$t.R \bullet t.S \neq \emptyset,$$
   where k is a constant of type $\gamma$, (or $\{\gamma\}$), $t.R$ and $t.S$ are component regions of type $\gamma$, (or $\{\gamma\}$).
   $\sigma_G(m)$ keeps the tuples of m for which condition $G$ is true.

3. **Geometric product** $\odot$:
   $\odot_{T \leftarrow R,S}(m)$ is only defined if $m$ has at least two geometric attributes $R$ and $S$. Its type is that of $m$ except that attributes $R$ and $S$ are replaced by a single geometric attribute $T$. The value of $\odot_{T \leftarrow R,S}(m)$ is defined as follows:

   $$m' = \{t' | t \in m,\ t'.T = t.R \bullet t.S \wedge t'.A = t.A \text{ for each non geometric attr. A in m }\}.$$

   As an example, consider $m$ of type :$\{[A, B, R, S]\}$, the product $m'$ is of type $\{[A, B, T]\}$, and is defined as:
   To each tuple $t$ of $m$ corresponds a tuple $t'$ in $m' = \odot_{T \leftarrow R,S}(m)$ such that:
   $$t'.A = t.A$$
   $$t'.B = t.B$$
   $$t'.T = t.R \bullet t.S.$$

We next illustrate these definitions with more involved operations that are expressible in the algebra just described:

1. **Geometric join** $\bowtie_G$:
   Given two maps $m_1$ and $m_2$ with respective geometric attributes $R_1$ and $R_2$, we have:

   $$m_1 \bowtie_G m_2 \stackrel{\text{def}}{=} \odot(\sigma_{R_1 \bullet R_2 \neq \emptyset}(m_1 \times m_2))$$

   Consider for example $m_1$ of type $\{[A, R_1]\}$, and $m_2$ of type $\{[B, R_2]\}$.
   $m = m_1 \bowtie_G m_2$ is of type $\{[A, B, R]\}$ and has for value:
   $m = \{t \mid t_1 \in m_1,\ t_2 \in m_2,\ t.A = t_1.A,\ t.B = t_2.B,\ t.R = t_1.R_1 \bullet t_2.R_2 \neq \emptyset\}$.

[3] One could also choose as built-in predicate $t \bullet R \neq \emptyset$.

2. **Cover $\mathcal{C}$:**
   As an example, consider the map representing the partition of France into districts. The cover of this map would include a single region (without alphanumeric data) representing the whole country without the inner borders between districts.

   Assume first each tuple of map $m$, whose cover we are looking for, has a single geometric attribute $R$ of type $\gamma$. Then $\mathcal{C}\ (m)$ is defined as follows:

   $$\mathcal{C}(m) = \sqcap \oplus \pi_R(m).$$

   Observe $\pi_R(m)$ is a projection of $m$ on the geometric attribute $R$. It is a map including a set of regions: by definition of the projection, it is of type $\{R : \gamma\}$. Obviously $\pi_R(m)$ represents a geometric object (region) as well, closed under the operations defined in Section 3.1.2. Applying to this region, the fusion operator $\oplus$, we get a single region. Applying to this region the singleton operator $\sqcap$, we indeed get a map which represents the cover of $m$. (More formally, to be consistent with the type of regions defined in Section 3.1.1, there should be a mapping between regions and named geometric components of a map).

   Let us now relax the assumption that $R$ should be of type $\gamma$. Assume indeed, that $R$ is of type $\{\gamma\}$. Then $\pi_R(m)$ would be of type $\{R : \{\gamma\}\}$. Then, $\sqcup_{\{\}}(\pi_R(m))$, where $\sqcup_{\{\}}$ is the set-collapse operator of the algebra of complex objects [AB88], represents indeed a region of type $\{\gamma\}$ and the cover is expressed as:

   $$\mathcal{C}(m) = \sqcap \oplus \sqcup_{\{\}}(\pi_R(m)).$$

   We shall see below that cover is useful for expressing the "superimposition" primitive of Section 2.

### 3.3 Back to the application of Section 2

We show in this section how the various user primitives of Section 2 can be expressed by means of the above operations on maps:

- **Projection** and **fusion** (Figure 2.2).
  Extracting the map of cereals, given the map $m$ of districts and cereals of type $\{[crops, districts, R]\}$ (Figure 2.2), is expressed by the following expression:

  $$\uplus\ Nest_{crops}\ \pi_{crops,R}\ (m).$$

  We first project out the district attribute, then we gather in a same tuple the regions having same crop value ($Nest$), and finally we apply the fusion operation.

- **Map overlay** (Figure 2.4) is trivially expressed through geometric join $\bowtie_G$.

- Similarly, **windowing** (Figure 2.7), i.e. selecting the set of regions intersecting with a given window $w$ is expressed by a geometric selection:

$$\sigma_{t.R \bullet w \neq \emptyset}(m),$$

where $t.R$ is the region value of tuple $t$ in map $m$. Observe that if $w$ is small enough, this allows to select a region (or a set of regions, if a map is not a partition of its cover) by windowing inside the region.

- **Clipping** (Figure 2.9), which consists in keeping from map $m$, only what is inside a given window $w$ is expressed by:

$$m \bowtie_G \sqcap w,$$

where $\sqcap w$ is the map with type $\{[R : \gamma]\}$ and with single tuple $[R : w]$.

- We end up with a more complicate operation: **superimposition** (Figure 2.5). This may be seen as:

  1. taking the cover of the map ($m_2$) that we want to superimpose onto a map ($m_1$).
  2. before superimposing $m_2$, we have to erase the location corresponding to the cover of $m_2$. In other words we have to make a "geometric difference" denoted $\ominus$ between $m_1$ and $m_2$ (we assume that $\mathcal{C}(m_1) \supset \mathcal{C}(m_2)$). We get map $m_3$.
  3. we finally superimpose $m_2$, i.e. take the (relational) union of $m_3$ and $m_2$

  In summary, superimposition is expressed as:

$$(m_1 \ominus m_2) \cup m_2$$

- Geometric difference can be expressed as (i) taking the difference of covers as a window, and (ii) clipping:

$$m_1 \ominus m_2 = m_1 \bowtie_G \sqcap(\mathcal{C}(m_1) - \mathcal{C}(m_2)).$$

# 4 The *apply* operator

To express the geometric operations on maps (Section 3.2), we only need (i) operations on regions, and (ii) a constructor called *apply*. It is denoted by: "$apply < f > (m)$". It takes as an input a map $m$ and gives as an output a map $m'$ obtained from $m$ by applying $f$ to each tuple of $m$.

The *apply* operator is not new. It is similar to the Lisp "Mapcar", and is an extension of the *replace* operator of the algebra of [AB88][4].

Let $m$ be a map of type $\{\tau\}$ and f a partial function of $\tau$ to $\tau'$. $f$ denotes a transformation to "apply" to each member of a set of a given type.
$apply < f > (m)$ is of type $\{\tau'\}$ and is defined as:

[4]The original name *replace* has been changed into *apply* because it sounded closer to the intuitive meaning of this constructor.

$$apply < f > (m) = \{f(t) | t \in m \wedge f(t) \text{ is defined}\}.$$

Informally speaking, if f is applied on a map $m$, $f$ can have other input parameters than $m$, for example $m_1, ..., m_n$, and is constructed from other apply functions and/or algebraic operations on $m, m_1, ..., m_n$. This operator is extremely powerful. In particular, it can express a variety of algebraic operations such as selection, projection, nest, unnest.

*apply* is just an extension of *replace* to take as specifications for the function $f$, *the operations of the algebra of regions*: $f$ is not only constructed from other apply specifications or algebraic operations on maps, but also from operations of the algebra of regions (Section 3.1.2). We show below that operations on maps such as geometric selection, fusion, join, can be expressed by an *apply* operation.

Let us first define more formally the construction of an *apply* specification. For a more complete definition, see [AB88].

1. **Basis for an apply specification:**
   if B is an attribute name in $\tau$, then B is an apply specification, and $B(t) \stackrel{\text{def}}{=} t.B$.

2. **Tuple construction:**
   if $AS_1, ..., AS_m$ are apply specifications from $\tau$ to $\tau_1, ..., \tau_m$, and $B_1, ..., B_m$ are attribute names, then $[B_1 : AS_1, ..., B_m : AS_m]$ is an apply specification from $\tau$ to $[B_1 : \tau_1, ..., B_n : \tau_n]$. Its effect is defined by:
   $[B_1 : AS_1, ..., B_m : AS_m](t) = [B_1 : AS_1(t), ..., B_m : AS_m(t)]$.

3. **Application of an operation:** although not necessary, we give separate constructions for operations of the algebra of maps and for operations of the algebra of regions.

   (a) if $op(M_1, ..., M_n)$ is an *algebraic operation* from $\tau_1$, ..., $\tau_n$ to $\tau$' and $AS_1, ..., AS_n$ are apply specifications from $\tau$ to $\tau_1$, ... $\tau_n$ then $op(AS_1, ..., AS_n)$ is an apply specification from $\tau$ to $\tau$'. Note that $\tau_i$, $\tau$' are types of any attribute (geometric or not).

   (b) if $op(r_1, ..., r_m)$ is an *operation on regions* from $\gamma_1, ..., \gamma_m$ to $\gamma$' (where $\gamma_i$, $\gamma$' are geometric types) and $AS_1, ..., AS_m$ are apply specifications from $\tau$ to $\gamma_1, ..., \gamma_m$ then $op(AS_1, ..., AS_m)$ is an apply specification from $\tau$ to $\gamma'$.

   In both cases, the effect of $op(AS_1, ..., AS_m)$ is defined by:
   $op(AS_1, ..., AS_m)$ (t) = $op(AS_1(t), ..., AS_m(t))$.

4. **Expression of conditional:**
   The effect is that of a selection. Again, we separate the case of regular selection from that of geometric selection.

   (a) if $AS_1, AS_2, AS_3$ are apply specifications from $\tau$ to $\tau_1, \tau_2, \tau_3$, then "if $AS_1 \theta AS_2$ then $AS_3$" is an apply specification from $\tau$ to $\tau_3$. It defines a partial function:

$$f(t) = \begin{cases} AS_3(t) & \text{if } AS_1(t) \theta AS_2(t) \\ undefined & \text{otherwise} \end{cases}$$

$\theta$ is one of the built-in predicates of the algebra on complex objects.

(b) if $AS_1, AS_2$ are apply specifications from $\tau$ to $\gamma, \tau'$, then "if $AS_1 \neq \emptyset$ then $AS_2$" is an apply specification from $\tau$ to $\tau'$. It defines a partial function:

$$f(t) = \begin{cases} AS_2(t) & \text{if } AS_1(t) \neq \emptyset \\ undefined & \text{otherwise} \end{cases}$$

**Proposition** [AB88]: the set of apply specifications is closed under composition. □

We end up this section in expressing the three main "geometric" operations on maps which have been described above (Section 3.2), through the following apply specifications:

- **Geometric product** $\odot$ :
  Let $m$ be a map of type $\{\tau\}$, where, without loss of generality, assume $\tau = [B_1 : \mu,\ B_2 : \mu,\ R_1 : \gamma,\ R_2 : \gamma]$. Then the geometric product $\odot_{T \leftarrow R_1, R_2}$ is expressed as:

  $$apply < [B_1 : B_1,\ B_2 : B_2,\ R : R_1 \bullet R_2] > (m).$$

  Indeed, it is an apply specification from $\tau$ to $\tau' = [B_1 : \mu,\ B_2 : \mu,\ R : \gamma]$, and can be seen as the tuple construction: $[B_1 : AS_1,\ B_2 : AS_2,\ R : AS_3 \bullet AS_4]$, where the $AS_i$ are the following apply specifications:

  $$AS_1 \equiv B_1$$
  $$AS_2 \equiv B_2$$
  $$AS_3 \equiv R_1$$
  $$AS_4 \equiv R_2.$$

  We denote the geometric product by $apply < \bullet > (m)$ (generic notation independent of the type of $m$).

- Similarly **fusion** $\uplus$ is denoted by $apply < \oplus > (m)$, and if m has the above type $\{\tau\}$, it is expressed as: $apply < [B_1 : B_1,\ B_2 : B_2,\ R : R_1 \oplus R_2] > (m)$.

- **Geometrical selection** $\sigma_G$**:**
  consider map $m$ of type $\tau$ with two geometric attributes $R$ and $S$ and the selection $\sigma_{t.R \bullet t.S \neq \emptyset}$; it is expressed as the following apply specifications:

  $$AS_1 \equiv R$$
  $$AS_2 \equiv S$$
  $$AS_3 = AS_1 \bullet AS_2$$
  $$if\ AS_3 \neq \emptyset\ then\ I_\tau$$

  where $I_\tau$ is the identity specification on $\tau$. Selection can then be specified as a composition of two *apply*'s: $apply < \neq \emptyset < \bullet > >$. If we had chosen $t.R \neq \emptyset$ as a built-in predicate for geometric selection (see footnote 4), selection would be specified as $apply < \neq \emptyset >$.

Likewise, one can define other apply specifications for the other operations of the section above, such as geometric join or cover. Consider the case of join: the **join** of $m_1$ and $m_2$ can be expressed as:

$$m_1 \bowtie_G m_2 \equiv apply <\neq \emptyset < \bullet >> (m_1 \times m_2).$$

We first compute the cartesian product of $m_1$ and $m_2$, then apply the "$\bullet$" operation, and finally check whether each tuple has a non-empty geometric component.

# 5 Redundant maps having same cover

A map is said to be in *normal form* if it has a single geometric attribute. Assume we are initially given a set of normal maps. Obviously, normal maps are not closed under the operations of Section 3.2.2: the cartesian product of two normal maps gives a map with two geometric attributes. This is why we did not require in the definition of maps (Section 3.2.1) that the number of geometric attributes in a map be limited to one.

Consider now a map $m_1$ of type $\{[A, R_1]\}$ and a tuple $t_1 = [A : a,\ R_1 : r_1]$ of $m_1$. Tuple $t_1$ is to be interpreted as follows: to each point $p$ of $r_1$ is associated the value $a$ for attribute $A$. Its "semantics" can thus be viewed as:

$$\{[a, p] \mid p \in r_1\}$$

Then, consider a map $m_2$ of type $\{[B, R_2]\}$ and the cartesian product $m_1 \times m_2$.

Tuple $t = [A : a,\ B : b,\ R_1 : r_1,\ R_2 : r_2]$ of $m_1 \times m_2$ can be interpreted as the set:

$$\{[a, b, p, q] \mid p \in r_1,\ q \in r_2\}$$

Cartesian products of normal maps are in general of poor interest (e.g. display couples of districts such that *district 1* has a city with more than one million of inhabitants and *corn* is grown in *district 2*).

Usually, given a tuple with two geometric attributes with values $r_1$ and $r_2$, one is interested in the geometric intersection $r_1 \bullet r_2$: given a tuple $[Crop : corn,\ R_1 : r_1]$ of map $m_1$ and a tuple $[District : d_1,\ R_2 : r_2]$ of map $m_2$, the information of interest is $[Crop : corn,\ District : d_1,\ R : r_1 \bullet r_2]$ which is one tuple of the *map overlay* $m = m_1 \bowtie_G m_2$.

Cartesian product $[Crop : corn,\ District : d_1,\ R_1 : r_1,\ R_2 : r_2]$ is only an intermediate step in the computation of map overlay:

$$\bowtie_G = \odot(\sigma_{R_1 \bullet R_2 \neq \emptyset}(m_1 \times m_2))$$

Observe that

$$\bowtie_G = \sigma_{R \neq \emptyset}(\odot(m_1 \times m_2)).$$

$\odot(m_1 \times m_2)$ is a normal map. Then, one may wonder whether it is not sufficient to keep all maps in normal form. This is possible if to each non normal map, we apply geometric product $\odot$.

In the above example, the tuple cartesian product $[A : a,\ B : b,\ R_1 : r_1,\ R_2 : r_2]$ would be replaced after having applied $\odot$ by: $[A : a,\ B : b,\ R : r_1 \bullet r_2]$ and would be interpreted as the **intersection** of regions $r_1, r_2$, i.e. as:

$$\{[a, b, p, p] \mid p \in r_1,\ p \in r_2\}$$

But such a "normalization" (applying $\odot$ to a non normal map) should lead to "equivalent" information:

Given $r_1 \bullet r_2$ for all tuples of $\odot(m_1 \times m_2)$, are we able to reconstruct $m_1$ and $m_2$, i.e. all values $r_1$ and $r_2$? The answer is of course no in the general case.

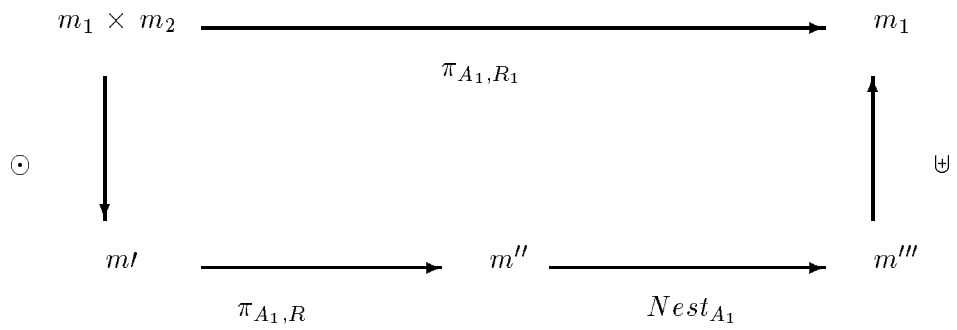
However, it turns out that for a very useful class of maps, $\odot$ is done *without loss of information*. The theorem below states that if $m_1$ and $m_2$ are normal maps with **identical cover**, then $\odot(m_1 \times m_2)$ is equivalent to $m_1 \times m_2$, i.e $m_1$ and $m_2$ can be reconstructed from $\odot(m_1 \times m_2)$.

For the sequel of this section, assume all maps $m$ are of type: $\{[R, A]\}$, where $A$ designates any complex object type on the non geometric attributes of $m$.

We further assume that a map, say with two geometric attributes $R_1$ and $R_2$, has been obtained at some point by applying cartesian product to two maps, say $m_1$ and $m_2$ in normal form and of types: $\{[R_1, A_1]\}$ and $\{[R_2, A_2]\}$.
$m = m_1 \times m_2$ is of type $\{[R_1, R_2, A_1, A_2]\}$.

**Theorem 5.1** *If $m_1$ and $m_2$ are two normal maps with* **same cover**, *then:*
$m_i = \uplus Nest_{A_i}(\pi_{A_i,R}(\odot(m_1 \times m_2))), i \in [1, 2]$,
*where $m' = \odot(m_1 \times m_2)$ is a normal map of type $\{[R, A_1, A_2]\}$.*

*This is illustrated in the following diagram, on the example of $m_1$.*




*Proof (Sketch)*:

Each tuple of $m'$ is of type $[R, A_1, A_2]$. By projection on $R$ and $A_1$ and nest on $A_1$, we get a map $m'''$ of type $\{[R_1 : \{R\}, A_1]\}$. Let us show that $\uplus m''' = m_1$.

To each tuple $t$ of $m'''$ corresponds a tuple $t_1$ in $m_1$ which has the same value for the non geometric attributes:
$\pi_{A_1}(m''') = \pi_{A_1}(m'') = \pi_{A_1}(m') = \pi_{A_1}(m_1 \times m_2) = \pi_{A_1}(m_1)$.

Let $[a_1, r_1]$ be the value of tuple $t_1$ in $m_1$ and $[a_1, \{s_1, ..., s_n\}]$ be the value of tuple $t$ in $m'''$.

By construction of $m'$, we have for all $i \in [1, n]$,

$$s_i = r_1 \bullet r_{2,i},$$

where $r_{2,i}$ is a region of map $m_2$ (we have for some tuple $t_2$ in $m_2$: $t_2.R_2 = r_{2,i}$).
$\{r_{2,1}, ...r_{2,n}\}$ is the set of regions of $m_2$.

Then applying $\uplus$ to $m'''$ implies applying $\oplus$ to the component $R$ of each tuple $t$ of $m'''$:
$\oplus t.R = \oplus\{s_1, ..., s_n\}$
$= \oplus\{r_1 \bullet r_{2,1}, ..., r_1 \bullet r_{2,n}\}$
$= r_1 \bullet \oplus\{r_{2,1}, ..., r_{2,n}\}$
$= r_1 \bullet \mathcal{C}(m_2)$
where $\mathcal{C}(m_2)$ is the cover of $m_2$. Since by definition of the cover of a map, $r_1 \subseteq \mathcal{C}(m_1)$ and since by assumption, both maps have same cover, then $r_1 \subseteq \mathcal{C}(m_2)$, and:

$$\oplus t.R = r_1 \bullet \mathcal{C}(m_2) = r_1.$$

Therefore, to each tuple $t_1$ in $m_1$ with value $[a_1, r_1]$, it corresponds a tuple $t$ in $m'''$ and

a tuple $t'$ in $\uplus m'''$ (of type $\{[R_1, A_1]\}$) such that $t'.R_1 = t_1.R_1 = r_1$.

Besides, by definition of $\uplus$, $t'.A_1 = t.A_1 = t_1.A_1 = a_1$, and there is no other tuple in $\uplus m'''$.

Thus, $\uplus m''' = m_1$.

The same argument holds for $m_2$.□

In conclusion, if the database includes only maps with same cover, maps with several geometric attributes (one for each component map) have redundant information. Keeping maps in normal form is sufficient for extracting any information about component maps.

# 6 Conclusion

We studied in this paper how to provide the designer of geographic databases with a database query language extensible and customizable to its own needs. For this, we took as an example a toy application on thematic maps and showed by using a complex objects algebra that application dependent operations could be expressed through a general construct called *apply*, which applies to each element of a set a user defined specific function.

The basic idea behind this construct is that it replaces a set of objects of type $\tau$ by a set of objects of type $\tau$' by applying to each object an application dependent (partially) defined function. The database query language itself is in charge of expressing application independent queries appropriate for manipulating a large amount of structured data. Provided the application is data driven and all user needs can be expressed through such a construct, this approach could be used for other applications such as full text systems, form management, etc.

The model we took in this paper was that of a complex object algebra [AB88]. A query language based on this model could be implemented with object oriented database systems such as Orion ([K+88, K+89]) or $O_2$ ([BBB+88, LRV88, LR89]). The design of database query languages (see for example [BCD89]) for such systems is still a research issue. Since one of the advantages of object oriented systems is customization and extensibility, such query languages should be good candidates for applications such as that investigated in this paper.

**Acknowledgments**
We are grateful to S. Abiteboul who read earlier drafts of this paper and suggested many improvements. The paper also benefits from discussions with C. Delobel and R. Jeansoulin.

# References


[AB88] S. Abiteboul and C. Beeri. On the power of languages for the manipulation of complex objects. Technical Report 846, INRIA, May 1988.

[AG87] S. Abiteboul and S. Grumbach. COL: A logic-based language for complex objects. Technical Report 714, INRIA, Septembre 1987.

[BB81] R. Barrera and A. Buchmann. Schema definition and query language for a geographical database system. In Hot Springs, editor, *IEEE Computer Architecture for Pattern Analysis and Image Database Management*, New York, Novembre 1981.

[BBB+88] F. Bancilhon, G. Barbedette, V. Benzaken, C. Delobel, S. Gamerman, C. Lécluse, P. Pfeffer, P. Richard, and F. Velez. The design and implementation of $O_2$, an object-oriented database system. Technical Report 20, GIP Altair, April 1988.

[BCD89] F. Bancilhon, S. Cluet, and C. Delobel. Query languages for object-oriented database systems: Analysis and a proposal. Technical report, GIP Altair (to appear), 1989.

[BK86] F. Bancilhon and S. Khoshafian. A calculus for complex objects. In *proc. ACM SIGACT-SIGMOD, Symp. on Principles of Database Systems*, 1986.

[CF80] N.S. Chang and K.S. Fu. A relational database system for images. In Chang and Fu, editors, *Pictorial Information Systems*. 288-321, Springer Verlag, 1980.

[CJ88] A.F Cardenas and T. Joseph. Picquery: A high level query language for pictorial database management. *IEEE Transactions on Software Engineering*, 14(5): pages 630–638, May 1988.

[CK81] S.K. Chang and T.L. Kunii. Pictorial database systems. *IEEE Transactions on Computer*, November 1981.

[Dav88] B. David. Le modèle Spatiarel. In *Quatrièmes Journées Bases de Données Avancées (BD3)*, pages 73–93, Bénodet, May 1988.

[Fra82] A. Frank. Map query: Data base query language for retrieval of geometric data and their graphical representation. *Computer Graphics*, 16, 1982.

[Fra84] A. U. Frank. Requirements for database systems suitable to manage large spatial databases. In *First International Symposium on Spatial Data Handling*, pages 38–60, Zurich, 1984.

[GCK+89] G. Gardarin, J.P. Cheiney, G. Kiernan, D. Pastre, and H. Stora. Managing complex objects in an extensible relational dbms. Technical report, INRIA, March 1989.

[Gut88] R.H. Guting. Geo-relational algebra : A model and query language for geometric database systems. In *Conference on Extending Database Technology (EDBT '88)*, pages 506–527, Venice, March 1988.

[Hul87] R. Hull. A survey of theoretical research on typed complex database objects. In J. Paredaens, editor, *Databases*, pages 193–256. Academic Press (London), 1987.

[HY84] R. Hull and C.K. Yap. The Format model: A theory of database organization. *ACM*, 31(3), 1984.

[K+88] W. Kim et al. Integrating an object-oriented programming system with a database system. In *Proc, 2nd Intl. Conf. on Object-Oriented Programming Systems, Languages and Applications*, San Diego, Septembre 1988.

[K+89] W. Kim et al. Features on the ORION object-oriented database system. In W. Kim and F. Lochovsky, editors, *Object-Oriented Concepts, Applications and Databases*. Addison-Wesley, 1989.

[KV85] G.M. Kuper and M.Y. Vardi. On the expressive power of the logical data model (extended abstract). In *proc. ACM SIGACT-SIGMOD, Int. Conf. on the Management of Data*, 1985.

[LM84] R.A. Lorie and A. Meier. Using a relational DBMS for geographical databases. In *Geo-Processing*, pages 243–257, 1984.

[LR89] C. Lécluse and P. Richard. Modeling complex structures. In *Object Oriented Database Systems, PODS*, Philadelphia, April 1989.

[LRV88] C. Lécluse, P. Richard, and F. Velez. $O_2$, an object-oriented data model. In *Conference on Extending Database Technology (EDBT '88)*, pages 556–563, Venice, March 1988.

[MOD87] F. Manola, J. Orenstein, and U. Dayal. Geographical information processing in Probe database system. In *International Symposium on Computer Assisted Cartography*, Baltimore, 1987.

[OM86] J. A. Orenstein and F. A. Manola. Toward a general spatial data model for an object-oriented data model. In *VLDB*, 1986.

[Ore86] J. A. Orenstein. Spatial query processing in an object-oriented database system. In *Proc. of the ACM SIGMOD*, pages 326–336, 1986.

[RFS88] N. Roussopoulos, C. Faloutsos, and T. Sellis. An efficient pictorial database system for PSQL. *IEEE Transactions on Software Engineering*, 14(5): pages 639–650, May 1988.

[Sam84] H. Samet. The quadtree and related hierarchical data structures. *Computing Surveys*, 16(2), June 1984.

[SDMO87] R. Sack-Davis, K.J. McDonell, and B.C. Ooi. GEOQL - a query language for geographic information system. In *Australian and New Zeland Association for the Advancement of Science Congress*, Townsville, Australia, August 1987.

[SMSE87] T. R. Smith, S. Menon, J.L. Star, and J.E. Estes. Requirements and principles for the implementation and construction of large-scale geographic information systems. *International Journal of Geographical Information Systems*, 1(1): pages 13–31, 1987.

[SR86] M. Stonebraker and L. A. Rowe. The design of POSTGRES. In *proc. ACM SIGACT-SIGMOD*, pages 340–355, 1986.

[SRG83] M. Stonebraker, B. Rubenstein, and A. Guttman. Application of abstract data types and abstract indices to cad data bases. In *Proc. of the ACM/IEEE Conf. on Engineering Design Applications*, pages 107–113, San Jose, 1983.

[SW86] H. J. Schek and W. Waterfeld. A database kernel system for geoscientific applications. In *Proc. of the Int. Symposium on Spatial Data Handling*, Seattle, Washington, 1986.