VII Simpósio Brasileiro de Geoinformática, Campos do Jordão, Brasil, 20-23 novembro 2005, INPE, p. 66-81.

Towards a generalized map algebra: principles and data types

Gilberto Câmara, Danilo Palomo, Ricardo Cartaxo Modesto de Souza, Olga Regina Fradico de Oliveira

Image Processing Division (DPI) – National Institute for Space Research (INPE)
Av dos Astronautas, 1758 – 12227-001 – São José dos Campos – SP – Brazil

{gilberto, danilo, cartaxo, olga}@dpi.inpe.br

Abstract. *Map Algebra is a collection of functions for handling continuous spatial data, which allows modeling of different problems and getting new information from the existing data. There is an established set of map algebra functions in the GIS literature, originally proposed by Dana Tomlin. However, the question whether his proposal is complete is still an open problem in GIScience. This paper describes the design of a map algebra that generalizes Tomlin's map algebra by incorporating topological and directional spatial predicates. Our proposal enables operations that are not directly expressible by Tomlin's proposal. One of the important results of our paper is to show that Tomlin's Map Algebra can be defined as an application of topological predicates to coverages. This paper points to a convergence between these two approaches and shows that it is possible to develop a foundational theory for GIScience where topological predicates are the heart of both object-based algebras and field-based algebras.*

1. Introduction

A *map* is useful metaphor for dealing with data in geographical information system (GIS) [Smith 1995; Egenhofer, Glasgow et al. 1999]. Of particular interest in GIS are *maps* associated to a continuous variable or to a categorical classification of space (e.g., soil maps). These types of maps are called 'coverages'[Frank, Volta et al. 1992; Volta and Egenhofer 1993; Erwig and Schneider 2000]. Different functions on coverages such as overlay and reclassification have been proposed in the literature, composing a set of procedures called '*Map Algebra'*. They allow the user to model different problems and to get new information from the existing data set [Berry 1987; Frank 1987; Güting 1988; Huang, Svensson et al. 1992].

The main contribution to map algebra comes from the work of Tomlin [1990]. Tomlin's model uses a single data type (a map), and defines three types of functions. *Local* functions involve matching locations in different map layers, as in "*classify as high risk all areas without vegetation with slope greater than 15%". Focal* functions involve proximal locations in the same layer, as in the expression "*calculate the local mean of the map values*". *Zonal* functions summarize values at locations in a layer contained in zones defined in another layer, as in the example "*given a map of city and a digital terrain model, calculate the mean altitude for each city."*

Tomlin's map algebra has become as a standard way of processing coverages, especially for multicriteria analysis. In recent years, several extensions to map algebra have been proposed. These include the GeoAlgebra of Takeyama and Couclelis [1997], an extension of map algebra that allows for flexible definitions of neighborhoods. Pullar [2001] developed MapScript, a language that allows control structures and dynamical models to be incorporated into map algebra. Ostlander [2004] suggests how map algebra could be embedded in a web service. Mennis et al. [2005] propose an extension of map algebra for spatio-temporal data handling. Frank [2005] discusses how map algebra can be formalized in a functional programming context and how this approach provides support both for spatial and spatio-temporal operations. Nevertheless, all extensions share the *ad hoc* nature of Tomlin's original proposal. They accept the foundations of Tomlin's algebra as a basis for their work.

Therefore, one of the open challenges in spatial information science is to develop a theoretical foundation for map algebra. We need to find out if Tomlin's map algebra can be part of a more general set of operations on coverages. We state these questions as: *"What is the theoretical foundation for map algebra?" "Could this theoretical foundation provide support for a more generic map algebra?"*

A second open issue involves the geometrical representations used in map algebra. Conceptually, map algebra could be implemented both on raster and vector representations [Timpf and Frank 1997]. In practice, however, we need topological and directional definitions suitable for using Tomlin's algebra with vector data. The absence of such definitions has limited map algebra implementations to use raster representations. This brings about a further question: *"How can we define a generalized map algebra suitable for both raster and vector representations?"*

To respond to these questions, we take the topological predicates of Egenhofer and Herring [1991] and the directional predicates of Frank [1992] as a basis for defining an algebra for coverages. Since these predicates are a good foundation for spatial query languages, it is natural to extend them on a map algebra context. Using these predicates, we propose an extended map algebra that includes Tomlin's as a subset. In what follows, we briefly review Tomlin's map algebra in section 2. In section 3, we present our set of map algebra operations. In section 4, we show some examples of its use, including some functions that are not expressible in Tomlin's work.

2. Map algebra: a brief review

2.1. Basic definitions

A map is a *function* $f: G \rightarrow A$, where:

1. The domain is a two-dimensional *geographical extent* $G \subseteq \Re^2$ which matches the planar coordinates of a geographical region in space. The extent G is partitioned into disjoint regions $r_1,..., r_n$, such $\cup\, r_i = G$.
2. The range is a set of *attribute values* whose domain is A.

For any 2D region $r \subset G$, a map returns a value $f(r) = a$, where $a \in A$. Note that this function knows how to calculate the value for any arbitrary region which is inside the area G. This definition of a map is general enough to include both numerical and categorical coverages defined in both vector and raster representations (here, we follow the OGC's definition of *coverage*; for more details, see [OGC 1998]).

There are two classes of functions in map algebra. *First order* functions take values as arguments (these are the functions associated to the map values). *Higher order functions* are functions that have other functions as arguments. Higher order functions are the basis for map algebra operations [Frank 1997]. Since maps are essentially first-order functions, operations on maps are applications of higher-order functions on all elements of a map. An example is "*classify as high risk all areas with slope greater than 15%*". To perform this operation, every element of a map is checked for the condition '*slope > 15%*' and those that satisfy it are labeled as *'high risk areas'*. The combination of the selection predicate and the classification function is the higher-order function.

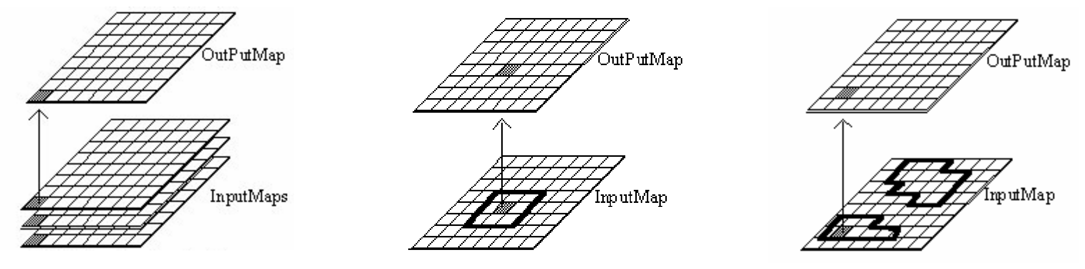
There are two ways to define higher-order functions for map algebra. The first is to define explicitly a special set of such functions. This is the approach taken by functional languages [Frank, 1977]. The second is to use implicit definitions. Functions can act has higher-order if they are applied to all elements of a map. The first approach is cleaner mathematically, but the second approach is more intuitive and has been more often used in practice. Examples of such functions include:

- *Single argument mathematical functions*: `log, exp, sin, cosine, tan, arcsin, arccosine, arctan, sinh, cosineh, tanh, arcsinh, arccosineh, arctanh, sqrt, power, mod, ceiling, floor.`
- *Single argument logical function:* `not.`
- *Multiargument functions*: `sum, product, and, or, maximum, minimum, mean, median, variety, majority, minority, ranking, count.`

2.2. Tomlin's map algebra

Tomlin's map algebra [1990] defines three types of higher-order functions for coverages. These functions apply a first-order function to all elements of map, according to different spatial restrictions**:**

- **Local functions**. The value of a location in the output map is computed from the values of the same location in one or more input maps. They include logical expressions such as "*classify as high risk all areas without vegetation with slope greater than 15%*" (Figure 1.a).
- **Focal functions**. The value of a location in the output map is computed from the values of the neighborhood of the same location in the input map. They include expressions such as "*calculate the local mean of the map values*" (Figure 1.b). Focal functions use the condition of adjacency, which matches the spatial predicate *touch.*
- **Zonal functions**: The value of a location in the output map is computed from the values of a spatial neighborhood of the same location in an input map. This neighborhood is a restriction on a second input map. They include expressions such as "*given a map of cities and a digital terrain model, calculate the mean altitude for each city*" (Figure 1.c). Zonal functions use the condition of topological containment, which matches the spatial predicate *inside*.



a. Local functions **b. Focal functions** **c. Zonal functions**

Figure 1. Tomlin's higher-order functions for map algebra (source: Tomlin [1990])

3. A generalized map algebra for coverages

This section presents a proposal that generalizes Tomlin's algebra. Our proposal only considers topological and directional relations between areas. A more complete proposal would also include other types of relations, such as line-area and line-line relations. However, we consider that a map algebra based on area-area predicates to be a useful tool for geographical analysis. The proposed map algebra has *nonspatial* and *spatial* higher-order functions. The former are equivalent to Tomlin's local functions and use single values in input maps. The latter generalizes Tomlin's focal and zonal functions using topological and directional predicates.

3.1. Conventions used in the paper

To define our map algebra, we need to define the data types and its functions. Types are a means of expressing abstractions in a computer language [Cardelli and Wegner 1985]. Types provide support for expressing abstractions by separating between specification and implementation. In the following definitions, we adopt some conventions:

1. Data types, functions and instances use `monospaced` font. Types are in **`boldface`** and their instances shown in `normal font`. For lists associate to types, we use **`list_typename.`**
2. Type definitions follow usual conventions for abstract data types. Types have an externally viewable set of functions and a set of axioms that are applicable to these functions [Guttag 1977].
3. Each function has a signature, given as

   `function:` **`typeA × typeB → out_type.`**

   `TypeA` and `typeB` are the types of the input parameters and `out_type` is the type of the output.
4. We describe each function in pseudocode. Instructions for flow control (e.g., **`for each`**) are in **`boldface`**. For attribution, we use '**`:=`**'. For equality test, we use **'`==`'** and the separator is ';'.

3.2. The map data type

Our basic data type is a `map`. Table 1 shows the definition of the `map` abstract data type. The definition assumes the existence of basic types `region`, `attr_domain` and `comparison`. They are, respectively, a 2D region, the domain of the attribute associated to the map and a comparison predicate.

Table 1. The map **data type**

Type **`map`** uses **`region, attr_domain`**	
	Instances used in the description
	`m:` **`map`** `r:` **`region`** `val:` **`attr_domain`**
Function	*Signature*
`getregions`	**`map`** → **`list_region`**
	Retrieve the list of regions of a map.
`contains`	**`map`** × **`region`** → **`bool`**
	Test if a map contains a region.
`overlaps`	**`map`** × **`region`** → **`bool`**
	Test if a region overlaps the extent of a map.
`insert`	**`map`** × **`region`** × **`attr_domain`** → **`map`**
	Given a value and region, insert the value in a map. The shorthand for this function is `m[r]:= val.`
`retrieve`	**`map`** × **`region`** → **`attr_domain`**
	Given a region, get its attribute value. The shorthand for this function is `val:= m[r].`
`new`	→ **`map`**
	Create a new map.
`add`	**`map`** × **`region`** → **`map`**
	Adds a region to a map.
`remove`	**`map`** × **`region`** → **`map`**
	Removes a region from a map.
Axioms	∪`(getregions (m)) ==` *G (the extent of the map)*
	The union of the regions of the map is its extent.
	`contains (m, r) == true iff r` ⊆ *G*
	Tests if a region is inside a map's extent.
	`insert(m,r,val) == error iff contains (m,r) == false`
	Inserting values into regions outside a map is impossible.

```
retrieve (m,r) == error iff contains (m,r) == false
```

Retrieving values from regions outside a map is impossible.

```
retrieve (insert (m, r, val), r) == value
```

What is inserted can be retrieved.

```
add (m,r) == error iff ((contains (m, r) == true) or
(overlaps (m,r) == true))
```

Planar enforcement rule: it is not possible to insert a new region whose limits are already partially or totally contained in a map.

```
remove (m,r) == error iff (contains (m,r) == false)
```

Only regions inside the map's extent can be removed from a map.

```
remove ((add (m, r), r) == add ((remove (m, r), r)
```

`Add` and `remove` are complementary.

To define the `map` data type, we also need to address an important problem in map algebra: finding the correspondence between regions in two maps. Given a region in the output map, a map algebra operation needs to find it in the input maps. Tomlin [1990] implicitly assumes there is an exact matching between the input and output maps. Although most implementations of map algebra follow this assumption, it is too strict. Instead, we consider the following assumptions about regions:

1. Each **`region`** knows its spatial reference system.
2. Given a **`region`**, a **`map`** can answer two questions about it. First, it can find out if the region is inside its geographical area. If this condition is true, it can calculate the value of its attribute function for that region.
3. Therefore, given a **`map`** `m` and a **`region`** `r` the function `m[r]` returns a value if and only if `contains (m,r)` is `true`.

These assumptions enable our proposed map algebra to handle different geometric representation and different scales and resolutions. Therefore, we can relax the assumption made by Tomlin [1990] there is an exact matching between the input and output maps.

3.3. Spatial predicates

As we show in section 2, spatial operations in Tomlin's map algebra use only two topological predicates (*'touch'* and *'inside'*). We use a more general set of spatial predicates. For topological predicates, we take the standard set {*'disjoint', 'equal', 'touch', 'inside', 'overlap', 'contains', 'intersects'*}, which cover all vector area-area relations. This set uses the 9-intersection model proposed by Egenhofer and Herring [1991], as adopted by the Open Geospatial consortium [OGC 1998]. For a rigorous definition for their application to raster representations, see [Winter 1995] and [Winter and Frank 2000].

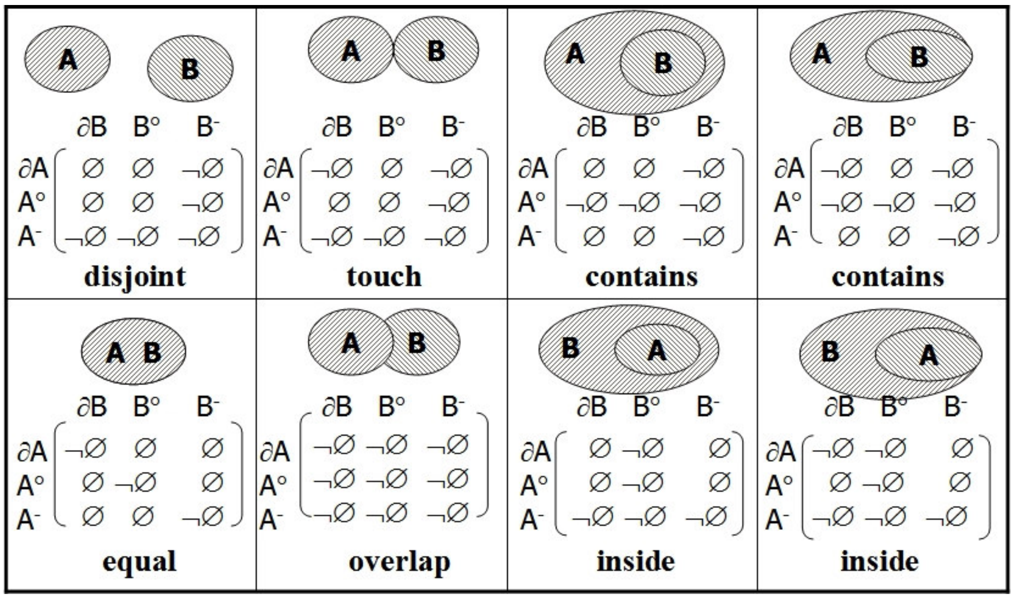


Figure 2. Topological predicates for area-area based on the 9-intersection matrix. Adapted from Egenhofer and Herring [1991] as adapted by the Open Geospatial consortium [OGC 1998]

The predicates in the minimal set *{'disjoint', 'equal', 'touch', 'inside', 'overlap'}* are mutually exclusive and cover all topological cases for area-area relations [Egenhofer and Herring 1991]. The predicate *'intersects'* is included for user convenience, following OGC's proposal [OGC 1998]., and '*intersects*' is defined as

intersects(a,b) $\Leftrightarrow$ *!disjoint(a,b)*

We also use the directional predicates *'north', 'south', 'east', 'west', 'center', 'northwest', 'northeast', 'southeast', 'southwest'*, originally proposed by Frank [1992]. Definition of these predicates for raster data is straightforward [Winter and Frank 2000] (see Figure 3). For area-area vector representation, Papadias and Egenhofer [1997] provide a set of suitable definitions for directional predicates. Figure 4 shows examples of application of semantic predicates.

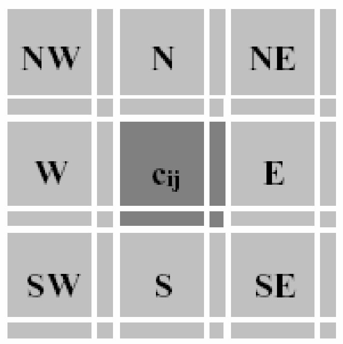


Figure 3. Directional predicates in raster representation. Source: Winter and Frank [2000]

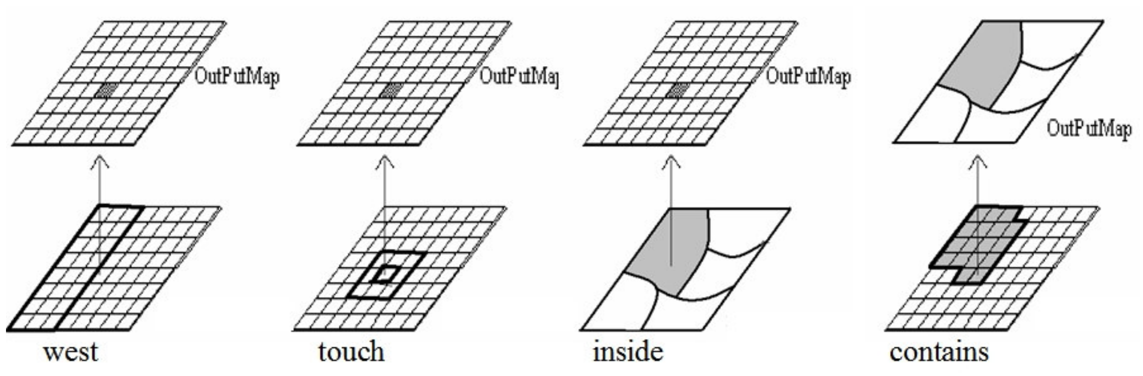


Figure 4. Examples of spatial predicates

3.4. Nonspatial operations

Nonspatial operations are higher-order functions that take one value for each input map and produce one value in the output map, using a first-order function as argument. They are the equivalent to Tomlin's local operations (see Figure 1.a). These operators include single argument functions, multiple argument functions, and a selection function, as described in Table 2.

Table 2. Nonspatial operators

Type **nonspatial_operation** uses **map, attr_domain, func_single, func_multiple, comp** (**comparison** is the type of comparison predicates, **func_single** is the type of single-varied functions, and **func_multiple** is the type of multivaried functions)		
Instances	m1, m2: **map** r: **region** value: **attr_domain** func1: **func_single**	maplist: **list_map** reglist : **list_region** valuelist: **list_attr_domain** funcn: **func_multiple**
Function	*Signature*	
single	**map × func_single × map → map** For all regions in the output map, find the value in the input map and calculate the output value:	

```
single (m1, func1, m2) {
   for each r in getregions (m2)
     if (contains (m1,r) == true)
        m2[r]:=  func1 ( m1[r] );
     else
        m2[r] := undefined;
}
```

`multiple` **`maplist × func_multiple × map → map`**

For all regions in the output map, find the values in the input maps and calculate the output value:

```
multiple (maplist, funcn, ml) {
  for each r in getregions (ml) {
    init valuelist;
    for each m in maplist {
     if (contains (m,r) == true)
         value := m[r];
     else
         value := undefined;
     add value to valuelist;
   }
   ml[r] := funcn (valuelist);
} }
```

`select` **`map × attr_domain × comp × map → map`**

For all regions in the output map, find the values in the input map and compare with a selected value. The resulting map will be a boolean map:

```
select (map1, value, comp, map2) {
for each r in getregions (map2) {
    if (contains (map1, r))
         map2[r]:= (map1[r] comp val);
    else
         map2[r]:= undefined;
}}
```

Nonspatial operations have a shorthand, as shown in Table 3. We also show some examples of nonspatial operations in Table 4.

Table 3. Convenience shorthand for nonspatial operators

```
output_map := single_arg_function (input_map);
// for single value functions

output_map := multi_arg_function (input_map_list);
// for multivalued mathematical functions

output_map:= map1 comp map2;
// for logical and comparison operators

output_map := input_map comp value;
// for selection operations
```

Table 4. Examples of nonspatial operators

Informal description	*Map Algebra Expression*
"Find the square root of the topography map"	`topoSqr := sqrt (topography);`
"Find the average of deforestation in the last two years"	`defAve := mean (defor2004, defor2003);`
"Select areas higher than 500 meters"	`high:= (topography > 500);`
"Select areas higher than 500 meters with temperatures lower than 10 degrees"	`high_cold := (topo > 500) and (temp < 10);`

3.5. Spatial operations

Spatial operations are higher-order functions that use a spatial predicate. These functions combine a selection function and a multivalued function, with two input maps (the *reference* map and the *value* map) and an output map. Spatial operations generalize Tomlin's focal and zonal operations and have two parts: *selection* and *composition*. For each location in the output map, the *selection function* applies a spatial predicate on the matching location in the input map. The result is a set of values that satisfy the predicate. The *composition function* uses the selected values to produce the result (Figure 5). The selection function takes a region in the output map and finds the matching region on the reference map. Then it applies the spatial predicate between the *reference* map and the *value* map and creates a set of values. These values are the input for the multivalued function. The implicit assumption is that the geographical area of the *output map* is the same as *reference map*. The spatial operation is described in Table 5.

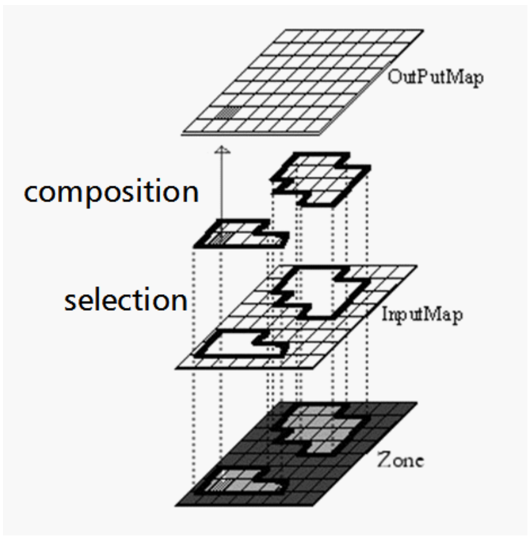


Figure 5. Spatial operations (selection + composition). Adapted from Tomlin [1990].

Table 5. Spatial operations

Type **spatial** uses **spat_pred, func_multiple, map** (**func_multiple** is the type of multivalued functions) (**spat_pred** is the type of spatial predicates)		
Instances	m1, refmap, m2 : **map**	r1, r2 : **region**
	v_list: **list_attr_domain**	func: **func_multiple**
	pred: **spat_pred**	
Function	*Signature*	
spatial	**map** × **map** × **pred** × **func_multiple** × **map** → **map**	
	Given a region in the *reference map*, it finds all regions in the value map that satisfy the spatial predicate. For all such regions, include the values of the attribute function in a list. The list is the input to a multivalued function which creates the result.	

```
spatial (m1, refmap, pred, func, m2) {
  for each r in getregions (refmap) {
     init v_list;
     for each r1 in getregions (m1) {
          if pred (r,r1)
               add m1[r1] to v_list;
     }
     m2 [r] = func (v_list);
} }
```

The spatial operations have a convenience notation shown in Table 6.

Table 6. Convenience notation for spatial operations

```
output_map:= multi_arg_function(value_map spat_pred ref_map);
```

The parameters for the spatial operations are:

- `output_map` is the map to be filled with the new values.
- `multi_arg_function` is a multiargument function as given in section 3.5.
- `value_map` is the map whose values are used to calculate the result.
- `spat_pred` is the spatial predicate (section 3.4).
- `ref_map` is the reference map used as a basis for applying the spatial predicate.

4. Examples and comparison with Tomlin's map algebra

In this section, we give some examples and compare our proposal to Tomlin's map algebra. First, we consider the operation: "*Find the local sum of regions in a deforestation map*". In Tomlin's algebra, this is a focal operation (Figure 1.b). Considering the deforestation map (`defor`) as input and the local sum map (`lsum`) as output, we state the operation as:

```
lsum := sum (defor touch lsum);
```

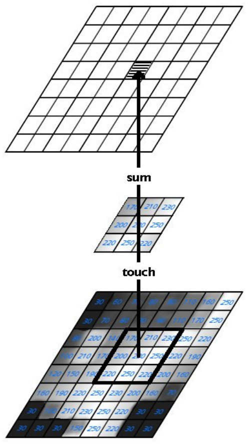


Figure 6. Local sum of deforestation

Note one interesting feature: the result (`lsum`) is also the reference map for the spatial predicate (`defor touch lsum`). This syntax may seem odd at first sight, but it follows from the generality of the proposal. By taking the reference map and the result map to be the same, we ensure the outcome satisfies the required condition ("local mean"). Had we used a third map as a reference, the result would be different if this map would not have the same spatial partitioning as the output map.

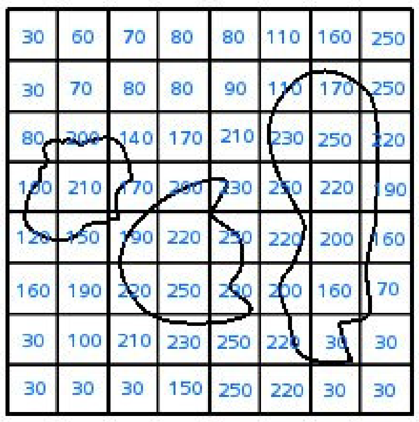
A second example would be the statement *"given a map of cities and a digital terrain model, find the average height of each city"*. This is a zonal operation in Tomlin's map algebra (Figure 1.c). Considering the digital terrain model (`dtm_map`) as the input map and the cities map (`cities_map`) as the reference, the operation is expressed as:

```
ave_city_height := average (dtm_map inside cities_map);
```

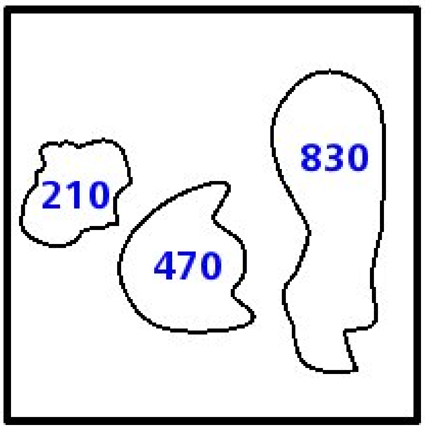
In this case, the output map will have the same spatial partitioning as the cities map. For each region in the output map, the algorithm finds the matching region in the cities map (which is trivial), and then applies the '*inside*' predicate between this region and all regions in the digital terrain model (DTM). The result will be a set of regions with associated values. Then, it applies a multivalued function (in this case, `average`) to get the result.

A third example is: "*Given a map of a native reservations and a deforestation map, find the sum of deforestation in the native reservations*". This is also a zonal operation in Tomlin's classification (Figure 1.c). Taking the map of native reservations (`reserves`) as the reference map and the deforestation map (`defor`) as the value map, we express the operation as:

```
deforRes := sum (defor inside reserves);
```



Deforestation map (grid) and native reservations (polygons)



Result (deforRes)

Figure 7. Calculation of deforestation in native reservations

The above examples show how the proposed map algebra expresses the spatial functions of Tomlin's map algebra, using *'touch'* and *'inside'*. Operators that use topological predicates other than *'touch'* and *'inside'* have no equivalent in Tomlin's algebra. The following are part of our map algebra and are not directly expressible by Tomlin's algebra: (a) operations involving the *'overlap', 'contains'* and *'intersects'* predicates; (b) operations involving directional predicates.

One example is the statement: "*Given a map containing a road and a deforestation map, calculate the mean of the deforestation along the road*". We have this input maps: deforestation (`defor`) and `road`. We select the areas that satisfy the condition of intersection and take the mean of the resulting values. These steps are illustrated in Figure 8. The expression is

```
defRoad := mean (defor intersects road);
```

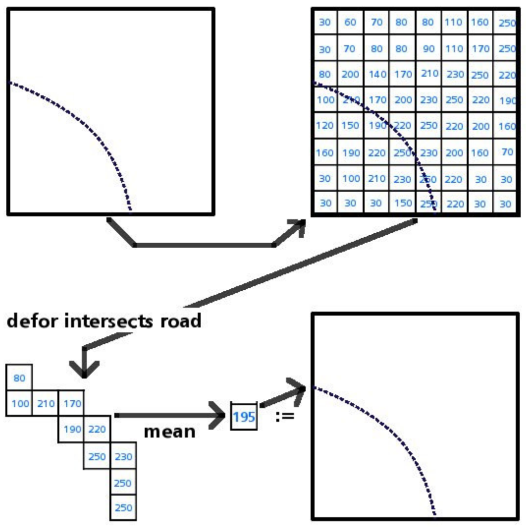


Figure 8. Calculation of the average deforestation along a oad

We present a comparison between the spatial operators and the focal and zonal expressions in Tomlin's map algebra in Table 7, where some operations have no direct equivalent in his algebra. In fairness to Tomlin, we should remember that his proposal mainly considers raster representations with the same spatial resolution, where some of these conditions would not be relevant. For example, the *'overlap'* predicate would not be applicable in his case.

Table 7. Comparison of spatial operators with Tomlin's map algebra

Informal Description	*Generalized Map Algebra*	*Tomlin*
"Local mean of topography"	`lmean:= mean (topo` **`touch`** `lmean)`	`FOCALMEAN OF TOPOGRAPHY`
"Given a map of cities and a topography map, calculate the mean altitude for each city."	`altcit:= mean (topo` **`inside`** `city)`	`ZONALMEAN OF TOPOGRAPHY WITHIN CITIES`
"Given a set of national forests, calculate the deforestation at the edges of each forest"	`defBord:= sum (defor` **`overlaps`** `forests)`	(no equivalent)
"Caculate the mean of the deforestation along the road"	`defRoad := mean (defor` **`intersects`** `road);`	(no equivalent)
"Find the deforestation to the north of the Amazon".	`defNorth:= sum (defor` **`north`** `amazon)`	(no equivalent)

5. Conclusions

This paper presents a generalized map algebra and compares it to the classical proposal by Tomlin. Our map algebra uses topological and directional spatial predicates. It expresses all Tomlin's algebra operations and enables operations that are not directly expressible by his proposal. Further generalizations of the proposed algebra are possible by involving the full set of topological operations. The next step in our work is to design, implement and validate a language that supports the proposed map algebra for spatial databases.

One of the important results of our paper is to show that Tomlin's Map Algebra can be seen as an application of topological predicates to coverages. Previously, topological operations on individual objects and those involving coverages have been dealt separately in GIScience theory. This paper points to a convergence between these two approaches and shows that it is possible to develop a foundational theory for GIScience where topological operations are the heart of both operations on objects and operations on fields. This theory would tie different types of spatial operations together into a single unified basis.

Acknowledgments

Gilberto Camara's work is partially funded by CNPq (grants PQ - 300557/19996-5 and 550250/2005-0) and FAPESP (grant 04/11012-0). Danilo Palomo's and Olga Oliveira's work is funded by CAPES. Gilberto Câmara would like to thank Andrew Frank for many stimulating discussions on the subjects of map algebra and functional programming.

References

Berry, J. K. (1987). "Fundamental Operations in Computer-Assisted Map Analysis." International Journal of Geographical Information Systems **1**(2): 119–136.

Cardelli, L. and P. Wegner (1985). "On Understanding Type, Data Abstraction, and Polymorphism." ACM Computing Surveys **17**(4): 471-552.

Egenhofer, M., J. Glasgow, O. Gunther, et al. (1999). "Progress in Computational Methods for Representing Geographic Concepts." International Journal of Geographical Information Science **13**(8): 775-796.

Egenhofer, M. and J. Herring (1991). Categorizing Binary Topological Relationships Between Regions, Lines, and Points in Geographic Databases. Orono, ME, Department of Surveying Engineering, University of Maine.

Erwig, M. and M. Schneider (2000). Formalization of Advanced Map Operations. 9th Int. Symp. on Spatial Data Handling Beijing, IGU.

Frank, A. (1987). Overlay Processing in Spatial Information Systems. AUTO-CARTO 8, Eighth International Symposium on Computer-Assisted Cartography. N. R. Chrisman. Baltimore, MD**:** 16-31.

Frank, A. (1992). "Qualitative Spatial Reasoning about Distances and Directions in Geographic Space." Journal of Visual Languages and Computing **3**(4): 343-371.

Frank, A. (1997). Higher order functions necessary for spatial theory development. Auto-Carto 13, Seattle, WA, ACSM/ASPRS.

Frank, A. (2005). Map Algebra Extended with Functors for Temporal Data. Perspectives in Conceptual Modeling: ER 2005 Workshops, Klagenfurt, Austria, Springer.

Frank, A., G. Volta and M. McGranaghan (1992). Formalization of Families of Categorical Coverages, University of Maine.

Güting, R. (1988). Geo-Relational Algebra: A Model and Query Language for Geometric Database Systems. Advances in Database Technology—EDBT '88 International Conference on Extending Database Technology, Venice, Italy. J. Schmidt, S. Ceri and M. Missikoff. New York, NY, Springer-Verlag. **303:** 506-527.

Guttag, J. (1977). "Abstract Data Types And The Development Of Data Structures." Communications of the ACM **20**(6): 396-404.

Huang, Z., P. Svensson and H. Hauska (1992). Solving Spatial Analysis Problems with GeoSAL, a Spatial Query Language. 6th Int. Working Conf. on Scientific and Statistical Database Management.

Mennis, J., R. Viger and D. Tomlin (2005). "Cubic Map Algebra Functions for Spatio-Temporal Analysis." Cartography and Geographic Information Science **32**(1): 17-32.

OGC (1998). OpenGIS Simple Features Specification for SQL. Boston, Open GIS Consortium.

OGC (1998). The OpenGIS Specification Model: The Coverage Type and Its Subtypes. Wayland, MA, Open Geospatial Consortium.

Ostländer, N. (2004). Interoperable Services for Web-Based Spatial Decision Support. 7th AGILE Conference on Geographic Information Science, Heraklion, Greece, AGILE.

Papadias, D. and M. Egenhofer (1997). "Hierarchical Spatial Reasoning about Direction Relations." GeoInformatica **1**(3): 251-273.

Pullar, D. (2001). "MapScript: A Map Algebra Programming Language Incorporating Neighborhood Analysis." GeoInformatica **5**(2): 145-163.

Smith, B. (1995). On Drawing Lines on a Map. Spatial Information Theory—A Theoretical Basis for GIS, International Conference COSIT '95, Semmering, Austria. A. Frank and W. Kuhn. Berlin, Springer Verlag. **988:** 475-484.

Takeyama, M. and H. Couclelis (1997). "Map Dynamics: Integrating Cellular Automata and GIS through Geo-Algebra." International Journal of Geographical Information Systems **11**(1): 73-91.

Timpf, S. and A. Frank (1997). Using hierarchical spatial data structures for hierarchical spatial reasoning. Spatial Information Theory - A Theoretical Basis for GIS (Int. Conference COSIT'97), Berlin, Springer-Verlag.

Tomlin, C. D. (1990). Geographic Information Systems and Cartographic Modeling. Englewood Cliffs, NJ, Prentice-Hall.

Volta, G. and M. Egenhofer (1993). Interaction with GIS Attribute Data Based on Categorical Coverages. Spatial Information Theory, European Conference COSIT '93, Marciana Marina, Elba Island, Italy. A. Frank and I. Campari. New York, NY, Springer-Verlag. **716:** 215-233.

Winter, S. (1995). Topological Relations between Discrete Regions. Advances in Spatial Databases—4th International Symposium, SSD '95, Portland, ME. M. Egenhofer and J. Herring. Berlin, Springer-Verlag. **951:** 310-327.

Winter, S. and A. Frank (2000). "Topology in Raster and Vector Representaion." GeoInformatica **4**(1): 35-65.