Map Algebra Extended with Functors for Temporal Data

Andrew U. Frank

Dept. of Geoinformation and Cartography,
Technical University, Vienna
frank@geoinfo.tuwien.ac.at

Abstract. This paper shows how to extend and generalize Tomlin's Map Algebra to apply uniformly for spatial, temporal, and spatio-temporal data. A specific data layer can be seen as a function from location to a value (Goodchild's geographic reality). Map layer but also time series and other similar constructions are functors, mapping local operations to layers, time series, etc. Tomlin's Focal Operations are mostly convolutions and the zonal operations are summaries for zones. The mathematical framework explained justifies polymorphic overloading of operation names like + are made to work for layers, time series, etc. There is also a uniform method to apply user-defined local functions to them. The result is a consistent extension of Map Algebra with a simplified user interface. The implementation covers raster operations and demonstrates the generality of the concept.

1 Introduction

The integration of temporal data into GIS is arguably the most important practical problem currently posed to the GIS research and development community (Frank 1998). Temporal data is collected for administration and scientific applications of GIS. Geographic data gives nearly always a snapshot of the state of our ever changing environment (Snodgrass 1992). These collections of snapshots contain information about changes and processes, but users are left to invent their own methods for temporal analysis. Many *ad hoc* extensions to commercial systems to handle spatial data from different epochs are reported (for example, at recent ESRI user conferences).

The central concept in GIS is the overlay process: Data from different sources are combined (figure 1). This is a computational version of the traditional physical overlaying of maps on a light table (McHarg 1969). Map Algebra is a strong conceptual framework for this method of spatial analysis and has changed little in the 20 years since its "invention" (Tomlin 1983b; Tomlin 1983a), which demonstrates its conceptual clarity. Dana Tomlin's Ph.D. thesis described Map Algebra in a semi-formal way providing all the information necessary for others to use and to produce implementations of Map Algebra (Tomlin 1983a). Several public domain or low cost implementations were around since 1980 (Tomlin's IBM PC version, OSUmap, IDRISI, to name but a few I have used). All the commercial GIS today organize geographic data in layers or themes (ESRI 1993) and contain map algebra operations;

OGC and ISO standards include them as well (sometimes overshadowed by a multitude of operations for the maintenance of the data and administrative queries).

I demonstrate in this paper that Map Algebra can be extended to include temporal data analysis. I start from the observation that processing of time series is similar to Map Algebra overlay operations. The clarification of the underlying theory—category theory and functors —leads to a generalization that extends Map Algebra to include processing of time series. Extended Map Algebra applies to spatial, temporal, and spatio-temporal data and generalizes the current implementations. It simplifies the user interface:

- The same operations apply to spatial, temporal, and spatio-temporal data, which reduces learning of commands, and makes the experience users have with Map Algebra valuable to solve spatio-temporal problems.
- Irrelevant detail is removed from the user interface.
- Users can define new functions without learning a special language.
- Map layers and snapshots are typed and errors are detected before starting lengthy processing to produce non-sense results.

The theory produces the consistency in the approach and justifies the solution. In addition, it gives guideline for the implementation and optimization of execution. The implementation in a very high-level language (Peyton Jones, Hughes et al. 1999) is only a few pages of code. It uses the second order concepts built into modern languages and demonstrates feasibility. The translation into imperative languages is straight forward.

The paper is structured as follows: the next section reviews map overlay and processing of time series. The following sections prepare the mathematical background, first discussing functions and then mappings between collections of functions seen as categories. The next two sections apply these concepts to extend processing of single values in formulae to local operations applied to collections of values, like map layer or time series. We then introduce summary functions and show, how focal operations fit. Zonal operations are a special case of summary operations and introduce comparable regional operations. At the end we list the improvement of Extended Map Algebra compared to current solutions and review the solution from an abstract point of view. The paper closes with a suggestion for future work and a summary.

2 Map Algebra

Map Algebra realizes the central tenet of GIS: the analysis of data related to the same location in space (figure 1). Tomlin has organized the operations with map layers in three groups (Tomlin 1990):

1. Local operations combine values from the same location,
2. Focal operations combine values from a location and the immediate surroundings, and
3. Zonal operations combine values from all locations in a zone.

In this paper I will first concentrate on the local operations and then show how focal and zonal operations fit in the framework.

Local operations in map algebra are intuitively understandable: the values of corresponding (homological) cells are combined to produce a new value (figure 1 left). The implementation is straightforward with loops over the indices of the raster array:

```
procedure overlay (a, b: layer; out result: layer; op:
function)
for i:= 1 to ymax do
     for j:= 1 to xmax do
          c [i,j] := op (a[i,j], b[i,j]);
```

The examples for map layers are male and female population per cell (*mpop* and *fpop*). They can be combined as shown in figure 1 left. Classification separates urban from rural areas, using a threshold of 300 persons/cell (figure 1 right).

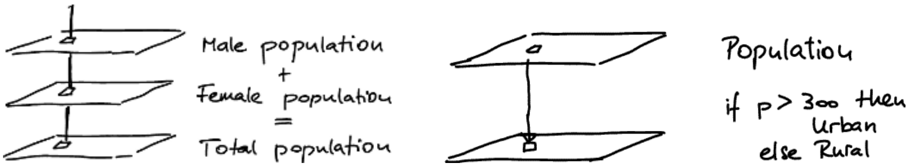

Fig. 1. (left) Total population is male population plus female population; (right) City areas are cells with total population higher than 300

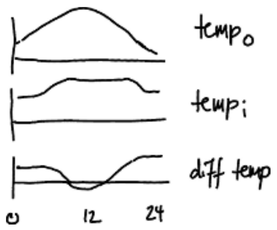

Fig. 2. The temperature difference between outside and inside

The processing of time series is similar (figure 2). Given two time series for the inside and outside temperature (*tempo* respectively *tempi*), the difference between the two at any given moment in time gives a new time series (*difftemp*). The computation for adding map layers or computing the difference between two time series is similar: combine homological, respective synchronous, values. What is the general rule? What are the limitations?

3 Computations are Functions Transforming Sets

A time series can be seen as a function *f (t)* = … Computation with functions is usual in electrical engineering; image processing uses a concept of images as a 2 dimensional function *g (x,y)* = ..(Horn 1986). Goodchild (1990; 1992) has suggested that geographic reality is a function of location and time *f (x, y, h, t)*. Can this viewpoint contribute to Map Algebra?

3.1 Definition of Function

"A function *f: S -> T* on a set *S* to a set *T* assigns to each $s \in S$ an element $f(s) \in T$. … The set *S* is called the domain of *f*, while *T* is the codomain. "... A function is often called a 'map' or a 'transformation' " (Mac Lane and Birkhoff 1967 p. 4). For example the function + takes pairs of values and maps them to a single value (+ :: (Int, Int) -> Int). The classification function *ur* takes a single value from the domain *population count* and maps it to the set *U* with the values *Urban* or *Rural.*

ur (p) = if p > 300 then Urban else Rural.

3.2 Application Functions

A function takes a value from a cell—or corresponding cells—and produces the value for the corresponding cell in the new layer. In figure 1 (left), this is the operation +, in figure 1 (right) the classification function *ur(p).* In figure 2 the two time series are combined with the function *diff (t1, t2) = t1 – t2.*

```
ur x = if x > (300::Float) then 'U' else 'R'
diff a b = a - b
```

These functions operate on values from domains of relevance to the user: Population density (in a discrete case: population count per cell), temperature, etc. They are sets of values and operations that map between them with rules like: $a + b = b + a$ (commutative law). The domains, the operations, and the rules are applicable to these operations form algebras (Couclelis and Gale 1986; Loeckx, Ehrich et al. 1996).

Users *compose* more complex formulae, for example to compute the percent difference between male and female population:

*(mpop – fpop) * 100/ pop*

or convert the temperature from degree Centigrade to degree Fahrenheit (*c2t*). For other datasets, one could compute the value for the 'universal soil loss formula'.

```
reldiff a b = abs (a - b) * (100.0::Float) / (a + b)
c2f t = 32 + (t * 9 / 5 )
f2c t = 5 * (t - 32)/ 9
```

3.3 Data Management Functions

The local operations for map layers or time series apply the application functions uniformly to every value (or pair, triple, etc. of homological values). This is encoded as a loop. Templates in C++ (Stroustrup 1991) separate the management of the data storage and access to the data from the processing with the application functions. The data management functions (in C++ called "iterators") are closely related to the data constructors.

A layer can be seen as a function that gives to each pair of indices the value at that position. For example, the layer population is a function:

pop :: (Int x Int) -> p.

In the code above these access functions are written as *x[i,j]*. The generalization to volumes and combinations of times series with spatial data is immediate. Comparable 1, 2 and 3 dimensional snapshot and temporal data constructors are:

timeSeries:: t -. v
layer :: xy -> v
stack of layer :: h -> xy -> v
volume :: xyh -> v
timeSeries of layer:: t -> xy -> v
timeSeries of volume :: t -> xy -> h -> v

4 Morphism

Functions (mappings, transformations) that preserve algebraic structure are called morphism; for example the function double is a morphism of addition. A morphism *M:: C -> D* maps the element of the domain to the codomain and maps the operation *f :: C -> C* on the domain to a corresponding operation *f':: D -> D* on the codomain, such that *M (f (x)) = f' (M (x))* (figure 3) (Mac Lane and Birkhoff 1967 p. 37)

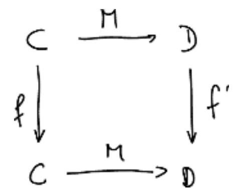
$$\begin{tikzcd} C \arrow[r, "M"] \arrow[d, "f"'] & D \arrow[d, "f'"] \\ C \arrow[r, "M"] & D \end{tikzcd}$$

Fig. 3. Morphism *M* maps *f* to *f'*

In many cases we use the same name for the two operations despite the fact that they apply to different domains (polymorphism). Calculation with logarithms using the rule *log (a * b) = log a + log b* provides an example, where an operation is mapped to a seemingly very different operation (* becomes +). The function *log* is a group morphism.

5 Construction of New Concepts from Existing Ones

The commonly available mathematical methods are not sufficient to construct a theory of map algebra. Functions operate on values, not layers. How to construct an extension?

5.1 Extension by Functors

Constructions in mathematics follow often the same pattern: A representation together with operations is found to be insufficient; the example from high school is that integers are insufficient to represent the solution to the problem of dividing 2 pies between 3 people, giving *2/3* to each.

Fractions are introduced as pairs of integers (numerator and denominator), such that the rules for addition and multiplication 'carry over'; integers map by the functor *F* to fractions by adding the denominator 1; rules like *a + 0 = a* are preserved: *a/1 + 0/1 = a/1*; addition and multiplication of fractions is commutative, etc. The same "trick" has been used to construct point coordinates, imaginary numbers, polynomials, etc. How to apply to geography?

"Many constructions of a new algebraic system from a given one also construct suitable morphism of the new algebraic system from morphism between the given ones. These constructions will be called 'functors' when they preserve identity morphism and composites of morphisms." *(Mac Lane and Birkhoff 1967 p.131).* Functors map domains to domains and morphism to morphism, i.e., they map categories to categories. Categories are sets of morphism, for example, the application domains used to encode map layers and the operations applicable to them form a category *V*. In our examples these are *integers*, *reals,* and the set *{Urban, Rural}* and the operations +, -, *, *ur, c2f*, etc. In each category, there is a special morphism, called *identity* morphism, which maps every value into the same value; it is the 'do nothing' operation. Categories have a single operation, namely *composition* of functions, written as '.' : *f (g (x)) = (f.g) x.*

6 Map Layers and Time Series are Functors

The representation of a snapshot of geographic reality for an area with a single number is insufficient. We may approximate the properties with a collection of values describing small areas (cells). A sampled image can be represented as a regular grid of values of type *v*, which we describe as a function

vlayer:: (Int x Int) -> v.

Local operations in map algebra correspond to the operations on values: we add two layers and understand this operation to add homological cell values. For example, adding two layers $C = A + B$ means *(c[i, j] = a [i,j] + b[i,j]).*

The constructor for map layers of a fixed size, which makes a "map layer of type *r*" from values of type *r*, is a functor *M*. The function *M,* which takes a single value *v* from the category *M* and maps it to a map layer produces layers with a all cells of this value (*a[i, j] = v*). Rules valid for the domain of cell values apply also for layers: for example the commutative law is valid for operations with layers: $A + B = B + A$). A similar argument—restricted to 1 dimension—shows that 'time series' is a functor as well (for historic reasons, it could be called *fluent* (Lifschitz 1990)). Indeed, all constructors listed above are functors.

6.1 Lifting Operation with Functors

Functors map operations on single values to operations on layers, time series, etc. It is customary to use the (polymorphic) operation *lift* to describe this mapping; we say the operation + is lifted to apply to layers (figure 1 left). If *M :: C -> D* is a functor and *op :: C -> C* is an operation then

op' = M (op) = lift op :: D -> D.

Because functors preserve composition (*M (f . g) = M f . M g*) not only single operations can be lifted, but also complex formulae like *ur, c2f,* or, e.g., the 'universal soil loss equation'.

Functors preserve the rules applicable to the operations. It is therefore appropriate to use the names of the ordinary functions directly for the corresponding operations on layers, time series, etc. (polymorphism). This simplifies the user interface: in stead of commands like *localSum, localDifference* (Tomlin 1990 p. 72) one uses directly the familiar +, - and writes:

```
pop = mpop + fpop
           -- adding male and female population
pop' = fpop + mpop
difftemp = tempi - tempo
           -- computing the difference in temp
tempif = lift c2f tempi
           -- conversion of time series to ºC
cities = lift ur pop
           -- classify in urban/rural
```

This works with time series of Population layers (e.g., population layers from 1950, 1960, 1970…). Applying the condition for *urban* to a time series of population layers gives a time series of a growing number of urban cells.

```
popTS = mpopTS + fpopTS
-- constructing a time series of total population
citiesTS = lift1 (lift1 ur) popTS
-- classify the time series of total population
```

If a formula contains a constant, the constant is lifted and becomes a layer with all values equal to the constant. Constant layers are useful to compute distances between locations: construct a layer where each cell contains the coordinate for its central location (*coord* layer) and then apply a "distance to point *p*" to it gives a layer with the distance of each cell from *p* (here *p= 2, 3*).

```
distance (x1,y1) (x2,y2)
           = sqrt (sqr (x1 - x2) + sqr (y1 - y2))
distTo23 = lift (distance (2,3)) coord
```

7 Summary Operations

For a time series, one might ask, what the maximum was or the minimum value, for example, for the temperature during a day, but the same questions is valid for a map: maximum or minimum population per cell, maximum or minimum height. For values on a nominal scale, one might ask what the most often occurring value is. From these primary summary values, other summary values are derived, e.g., average or higher statistical moments.

There are only a small number of functions that can serve to compute a summary. I propose the hypothesis, that only binary functions $f :: a \to a \to a$ with a zero such that $f(0, a) = a$ and which are additive can be used. Additive for a function in this context means that for two datasets A and B, the summary of the summary $(f(A))$of A and the summary $(f(B))$of B must be the same as the summary of A merged with B $(A + B)$

$$f(f(A), f(B)) = f(A + B).$$

A summary operation has no corresponding operation on a single value; a summary with the + operation is comparable to an integral over the time series, the map layer or the data volume. The operation *fold* converts a summary function like + to an operation on a data collection; for example *fold (+)* computes the sum for the data collection, *fold min* the minimal value. The second order function fold depends only on the data type.

Summary operations can apply to the complete data collection (e.g., a time series of data volume) or be applied to 'slices', for example the summary for each data cube in a time series gives a time series of single (summary) values (figure 4). Another example: a summary in a data cube can be for each layer giving a value for each height, or can be vertical for each location, summing all the values in the different heights.

```
popsum = sum pop           -- total population
tempimax = fold max tempi -- max temperature
tempomin = fold min tempo
popTSsum = lift sum popTS
            -- time series of total population
```

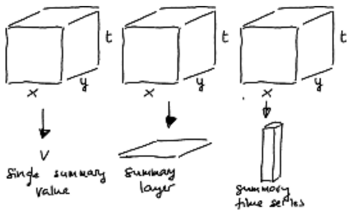

Fig. 4. Different summaries for a time series of layers

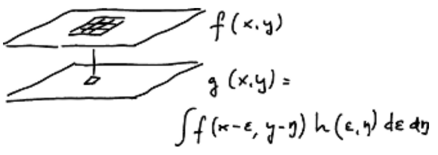

Fig. 5. A focal operation combines values in a neighborhood

8 Focal Operations in Map Algebra: Convolutions

Focal operations combine values in the neighborhood of a value to a new value (figure 5). Such operations do not have an equivalent operation for single values. Considering a layer as function reveals that Tomlin's zonal operations (or nearly all of them) are convolutions. A convolution of a function *f* with a weight function *h* (kernel) is the integral of the product of the layer *f* times the kernel function *h* (this is comparable to a weighted average).

Focal operations can be applied to functions of any number of variables. The 2-, 3-, or 4- dimensional convolutions are constructed analogously as double, triple, and quadruple integrals. Convolution is linear and shift-invariant (Horn 1986 p.104). Best known are the discrete forms of convolution, used as filter operations in image processing software. A 'focal average' operation is close to a filter to smooth an image, with a special weighting function (a Gaussian), but convolution can be used to detect edges, etc.

Applying a convolution to a multi-dimensional dataset, say a data volume (*vol: z -> x -> y -> v*) allows some options, which must be selected according to the application requirements: A *3d*-convolution can be applied to the volume (*conv k3 v*) (figure 6 left). Considering the data volume as a stack of layers, a *2d*-convolution can be applied to each layer in the stack (*lift (conv k2) v*) (figure 6 middle). Applying a *1d*-convolution to the data in each vertical data set is *conv k1 v* (where *k1* is a *1-d* kernel to select by polymorphism the *1d* convolution, which is then applied as a local operation to homologous values in the stack) (figure 6 right). In each case, different neighborhoods are used in the convolution.

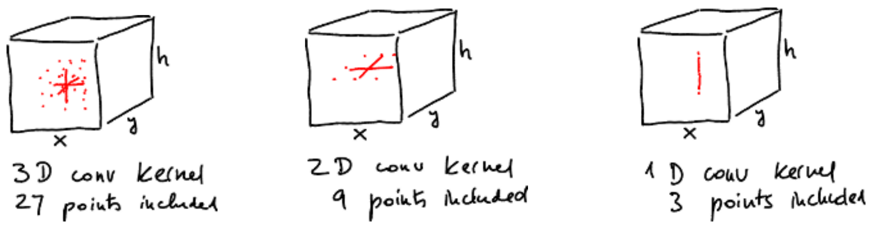

Fig. 6. 3-, 2-, and 1-dim convolution

Focal operations are very flexible if we permit functions that include conditions. For example, the Game of Life of John Conway can be expressed as a convolution or a convolution can be used to compute the next step in a simulation of a forest fire, given wind, amount of fuel, etc. The kernel could be the rule: if the central cell contains fuel and is burning then it continues to burn (with reduced fuel left); if it is not burning and one of the neighbor cells upwind from this cell then this cell starts burning as well.

9 Zonal Operations in Map Algebra

For time series, it is often desirable to compute the summary values for each day; for example a daily maximum and minimum. Such summaries for a regular aggregation are rare for space; here the summary should be computed for an irregular area. The area can be given as a spatial layer with value *True* for the cells included in the area; this is in image processing known as the characteristic function. Combining this area layer with the value layer by a function *f (a, v) = if a then v else 0* and then apply the summary function to the result gives the summary for the desired area. This works for all dimensions of data sets. The combination of the layers and the summary operation can always be combined in a single operation (for a proof (Bird and de Moor 1997 p. 10)).

Tomlin introduced summary functions that compute a single a summary value for a zone. Tomlin's definition of a zone is an "area which has some particular quality that distinguishes it from other geographic areas" (Tomlin 1990 p. 10). Practically, zones are areas that have the same value in a layer; zones are not necessarily connected.

For many applications of spatial analysis zones are not suitable and simply connected regions are required. For example, to assess the suitability of a habitat the size of a connected wood area (a wood region) can be important and cannot be replaced with the size the wood zone, which gives the total wood area in the area (Church, Gerrard et al. 2003). To identify regions requires a function to form the connected components of a zone.

Zonal operations compute for each zone (or for each region) a summary value. In Tomlin's Map Algebra, the zonal operations are defined such that they produce again a map layer, where every cell in the layer has the value obtained for the zone (or region) it is part of. This is necessary to make Map Algebra closed—operations have layers as inputs and produce layers as results.

The zonal operations apply to all dimensions of data collections. For time series, one might ask: what is the maximum temperature for the 'rain zone'? (The 'rain zone' can be defined, for example, as days for which more than 5 mm precipitation was measured).

```
condUrban uOr popVal = if uOr == 'U' then popVal else
zero
popCity = lift condUrban cities pop
-- restrict the population values to the urban zone
citypop = sum popCity
-- total population in urban zone

-- the urban zones and the their total population
--          for the population time series
popCityTS = lift (lift condUrban) citiesTS popTS
citypopTS = lift sum popCityTS
```

10 What Is Achieved?

10.1 Consistent Operations for Maps, Time Series, Stacks of Maps, etc.

Map layers and time series, their combination and their extensions are all functors and instances of a single concept, namely a class of functors *layerN*, where *n* can be 1 for time series, 2 for Tomlin's Map Algebra, and 3 for either *2d* space and time or *3d* space and 4 for *3d* space and time (figure 6). This covers Godchild's concept of 'geographic reality' *f (x,y,h,t)* and reveals it as a functor.

10.2 Polymorphic Application gives Local Operations "for Free"

Polymorphism and automatic lifting of functions makes operations and functions that apply to single values to apply to data collections as local operations. In stead learning a command like *localSum* or *localMinimum* one can directly use the operations + or *min* and write l*ayerC = layerA + layerB* or *layerD = min (layerA, layerB).* Functors preserve the rules applicable to operations; users know that $C = A + B$ is the same as $C' = B + A$.

10.3 Extensibility Without Special Language

If an application requires the value for a formula, for example, how much is the difference between the highest and the lowest value in relation to the average, one could search in the manual if such a Map Algebra function exists or combine it from available functions, computing intermediate layers. Knowing that layer is a functor, we just write the function and apply it (with *lift*) to the layers. This is valuable, as sequences of routine Map Algebra operations can be combined in a single formula and users need not learn a special scripting language.

```
r a b = 2.0 * (max a b - min a b )/(a + b)
rpop = lift r mpop fpop
```

10.4 Typing

The description of map algebra given here is compatible with a typed formalism (Cardelli and Wegner 1985; Cardelli 1997) and requires it to make polymorphism work. It is therefore possible to identify errors and advise users to avoid non-sense operations: if A is a layer with values *True* or *False*, then $C = A + B$ is ill typed. The operation + is not applicable to layers with Boolean values, because + is not applicable to Boolean values and can not be lifted to layers of Booleans.

10.5 Implementation

The strong theory helps us with the implementation. In a modern language, constructs like Functor or lift are directly expressible (in Haskell directly, in C++ with templates).This gives an implementation for local operations of map algebra in few lines of high level code.

The use of the GHCi (available from www.haskell.org) interpreter gives a full environment, in which functions can be loaded from files and computations executed interactively and results displayed. All the examples included here have been run in this environment. Functions that are used often could be compiled with GHC to improve performance. The concepts explained here are not tied to this or any other programming languages and can be implemented with any current programming language.

10.6 Optimization

Using Map Algebra leads to the computation of many intermediate layers, which are then used as inputs to other operations. This is useful when exploring a question; once a sequence of operations that lead to the desired result is found, the intermediate layers are a nuisance, because they require careful naming to avoid confusions. In the categorical framework, the production of intermediate layers can be avoided and all the operations done in one sweep of the data, for example using rules like:

lift f (lift g la lb) = lift (f . g) la lb.

This rule should remind programmers of methods to merge sequences of loops into a single one! Having a strong theory gives us guidelines when and how optimizations are possible; understanding when optimization is not permitted may be more important to avoid producing wrong results. For example, it is not evident how to combine local and focal operations; understanding that a focal operation is a convolution and as such linear answers the question.

11 Why Does This Work

Category theory is the abstraction from algebra, it is 'one step up the abstraction ladder' (Frank 1999).Why is it useful here? Category theory is unifying very large parts of mathematics and brings them into a single context. It integrates in the same framework the theory of computing: "The main methodological connection between programming language theory and category theory is the fact that both are essentially 'theories of functions'." (Asperti and Longo 1991 p.ix).

All computing is in a single category, namely the category of sets with functions. The state of a computer is the collection of the values in all memory cells, CPU, output channels, etc. All computing is a—program controlled—transformation of the state of the machine to a new state. Every program, but also every step in a program is a function that transforms the state of the machine; application programs are composition of functions.

In the categorical framework, a mathematical treatment of computing becomes feasible. Besides standard mathematics, the construction of data types and the control of flow in a program with its statements are expressible. Functions with *if_then_else* logic can be written with the McCarthy conditional function and compose with other functions.

In general, programming an application means—independent if the analyst and programmer understand this theoretically or not—as the formalization of a suitable category for the concepts and operations of the application and then a mapping of this category to the category of sets because this is the category in which the program works.

12 Future Work

The description here as well as Tomlin's original text, does not depend on a discrete, regular raster. The theory is developed in terms of continuous functions in 2, 3, or more variables. The translation to a discrete, regular raster based implementation is immediate; the use of data represented by irregular subdivision (so-called vector data) is more involved. I suggest as future work a comprehensive investigation resulting in a theoretically justified implementation of Extended Map Algebra covering both regular and irregular subdivision data with the same operations.

13 Conclusion

As expected, there exists a strong theory behind Map Algebra. The identification of this theory has lead us to see commonalities between the methods we use to process spatial and temporal data and to understand that the general concept of map algebra can be applied to spatial data of 2 or 3 dimension, to temporal data, and to spatio-temporal data of $2d + t$ or $3d + t$; it can be used to combine data collections of any dimension.

The formal theory gives us rules for the reorganization of processing geographic data with Map Algebra operations. Descriptions of processing steps can be formalized and optimized using rules that guarantee that the same result is obtained with less effort.

The user interface can be reduced; the number of operation names required is drastically reduced exploiting polymorphism when it is appropriate. This should reduce the instruction time necessary for future users of GIS. The introduction of typed data can advise users when they try to compute nonsensical operations ahead of time (Cardelli 1997).

My esteemed PhD advisor Rudolf Conzett used to quote Boltzmann: "There is nothing more practical than a good theory". I believe that formal investigations lead to a deeper understanding and, as a consequence, a simplified solution. This paper demonstrates how theory leads to more powerful Extended Map Algebra with at the same time a simplified user interface.

Acknowledgements

Many of my colleagues and students have helped me to advance to this point. Particular thanks go to the Haskell community (www.haskell.org) for their continuous efforts to produce tools to play with category theory and apply it to practical problems. Funding from the European Commission for projects DigitalSEE, Geocompass, Georama and REVIGIS is acknowledged. Special thanks to my former student Dorenbeck for his work on an MSc. Thesis "Spatial Algebras in Geographic Information Systems" in 1991. The discussions with Max Egenhofer, Werner Kuhn and Gilberto Camara since then have contributed substantially to my current understanding.

References

Asperti, A. and G. Longo (1991). Categories, Types and Structures - An Introduction to Category Theory for the Working Computer Scientist. Cambridge, Mass., The MIT Press.

Bird, R. and O. de Moor (1997). Algebra of Programming. London, Prentice Hall Europe.

Cardelli, L. (1997). Type Systems. Handbook of Computer Science and Engineering. A. B. Tucker, CRC Press**:** 2208-2236.

Cardelli, L. and P. Wegner (1985). "On Understanding Types, Data Abstraction, and Polymorphism." ACM Computing Surveys **17**(4): 471 - 522.

Church, R. L., R. A. Gerrard, et al. (2003). "Constructing Cell-Based Habitat Patches Useful in Conservation Planning." Annals of the Association of American Geographers **93**(4): 814-827.

Couclelis, H. and N. Gale (1986). "Space and Spaces." Geografiska Annaler **68**(1): 1-12.

ESRI (1993). Understanding GIS - The ARC/INFO Method. Harlow, Longman; The Bath Press.

Frank, A. U. (1998). GIS for Politics. GIS Planet '98, Lisbon, Portugal (9 - 11 Sept. 1998), IMERSIV.

Frank, A. U. (1999). One Step up the Abstraction Ladder: Combining Algebras - From Functional Pieces to a Whole. Spatial Information Theory - Cognitive and Computational Foundations of Geographic Information Science (Int. Conference COSIT'99, Stade, Germany). C. Freksa and D. M. Mark. Berlin, Springer-Verlag. **1661:** 95-107.

Goodchild, M. F. (1990). A Geographical Perspective on Spatial Data Models. GIS Design Models and Functionality, Leicester, Midlands Regional Research Laboratory.

Goodchild, M. F. (1992). "Geographical Data Modeling." Computers and Geosciences **18**(4): 401- 408.

Horn, B. K. P. (1986). Robot Vision. Cambridge, Mass, MIT Press.

Lifschitz, V., Ed. (1990). Formalizing Common Sense - Papers by John McCarthy. Norwood, NJ, Ablex Publishing.

Loeckx, J., H.-D. Ehrich, et al. (1996). Specification of Abstract Data Types. Chichester, UK and Stuttgart, John Wiley and B.G. Teubner.

Mac Lane, S. and G. Birkhoff (1967). Algebra. New York, Macmillan.

McHarg, I. (1969). Design with Nature, Natural History Press.

Peyton Jones, S., J. Hughes, et al. (1999). Haskell 98: A Non-strict, Purely Functional Language.

Snodgrass, R. T. (1992). Temporal Databases. Theories and Methods of Spatio-Temporal Reasoning in Geographic Space (Int. Conference GIS - From Space to Territory, Pisa, Italy). A. U. Frank, I. Campari and U. Formentini. Berlin, Springer-Verlag. **639:** 22-64.

Stroustrup, B. (1991). The C++ Programming Language. Reading, Mass., Addison-Wesley.

Tomlin, C. D. (1983a). Digital Cartographic Modeling Techniques in Environmental Planning, Yale Graduate School, Division of Forestry and Environmental Studies.

Tomlin, C. D. (1983b). A Map Algebra. Harvard Computer Graphics Conference, Cambridge, Mass.

Tomlin, C. D. (1990). Geographic Information Systems and Cartographic Modeling. New York, Prentice Hall.