Algebraic Optimization of Combined Overlay Operations[*]

Claus Dorenbeck[†]
Max J. Egenhofer[‡]
National Center for Geographic Information and Analysis
University of Maine
Orono, ME 04469, U.S.A.
CLAUS@MECAN1.bitnet
MAX@MECAN1.bitnet

Abstract

The operations necessary to combine map layers are formalized with algebraic specifications. This shows that arithmetic operations upon discrete spatial subdivisions are reduced to a single, parametric overlay operation, the actual behavior of which is determined by a value operation which combines the non-spatial attributes of the individual cells of the corresponding layers. The novel approach is the application of these formalisms to find more efficient strategies for processing several overlay operations at an implementation-independent level. Two particular strategies are investigated: (1) the elimination of equivalent subexpressions to reduce the complexity of the overlay operation and (2) the integration of several overlay operations into a single one.

1 Introduction

Spatial data models (Peuquet 1984, White 1984, Frank and Kuhn 1986, Herring 1987, Egenhofer *et al.* 1989) and spatial data structures (Peucker and Chrisman 1975, Corbett 1979, Nagy and Wagle 1979, Samet 1989) have been extensively studied in the past. More recently, the interest in the relations between spatial data models and spatial data structures has increased (Egenhofer and Herring 1991, Frank 1991b, Frank and Mark 1991, Goodchild 1991). Initial results of these investigations are:

- A *spatial data model* is the formalization of *spatial concepts* so that they can be represented in a computer.

[*]This work was partially funded by a grants from NSF under grant number SES 88-10917 and Intergraph Corporation, principal investigator Andrew U. Frank.

[†]On a leave of absence from the University of Bremen, F.R. Germany.

[‡]Additional support from Digital Equipment Corporation under grant number TP-765536 is gratefully acknowledged.

- A *spatial data structure* is the implementation of a particular spatial data model.
- A spatial data model may have multiple implementations with various effects on the performance and the storage requirements.
- A spatial data structure must fulfill the properties of the operations specified for the data model, in order to be a valid implementation of a spatial data model.

For example, the data model of a regular subdivision of space into squares of equal size, frequently called a *raster model*, may be implemented with different spatial data structures, such as a 2-dimensional array, run-length encoded, as a quadtree data structure, etc. (Samet 1989).

Within this framework of spatial data models and spatial data structures, a question of particular interest is, "How to describe formally the behavior of the operations?" This question covers the properties of the operations, i.e., the linkage between a data model and its various implementations, but, more importantly, also the specifications of the properties of their combinations. Note that it does not treat the actual implementation, i.e., a particular data structure or an algorithm.

Formalizations of spatial data models and GIS operations are necessary to (1) verify that an implementation, i.e., a spatial data structure, does what was set forth by the spatial data model, and (2) compare the semantics of different spatial data models (Frank 1987, Smith and Frank 1990). Non-spatial data models have been formalized, for example, by the relational algebra which specifies the behavior of the operations upon relational tables (Codd 1970, Ullman 1982), but only subsets of spatial algebras exist, e.g., for topological relationships (Egenhofer and Herring 1990) or directions (Peuquet and Ci-Xiang 1987, Chang *et al.* 1989, Frank 1991a). Each of these approaches is limited to a very specific class of operations and no attempts have been made to integrate them into a larger system.

The Map Analysis Package provided the first comprehensive collection of analytical and spatial operations on the basis of regular tessellations (Tomlin 1983, Tomlin 1990). It describes map overlay operations informally, without applying the mathematical rigor necessary to analyze the behavior of the operations, leaving ample space for ambiguous interpretations. One implementation of this *MAP algebra* describes formally these operations in the C^{++} programming language (Chan and White 1987), but lacks the definitions of the corresponding observe operations so that no axioms about the *behavior* of the operations can be formulated.

More formal approaches are based on the Euclidean plane and the representation of spatial data in terms of points, lines, and areas. A formalisation of the non set-theoretic part of Euclidean geometry (Tarski 1959) gives a collection of thirteen elementary axioms. An algebraic specification of graphic data types formally defines the semantics of a simple graphics programming language without geometric operations (Mallgren 1982). An algebra for geometric constructions, based upon well-known algebraic structures such as rings and fields (Goguen 1989), demonstrates the use of algebraic specifications for spatial objects, however, it is limited to a few, very basic constructs in plane geometry. The geo-relational algebra enhances the relational model with computational geometry operations (Güting 1988). A formalized interpretation of operations upon maps uses set operations and a construct similar to constructs in functional programming languages, such as *mapcar* in LISP, applying a user-defined function to each element of a set (Scholl and Voisard 1989).

This paper addresses the formalization of operations on regular tessellations to assess optimization strategies, particularly for combinations of overlays. The operations necessary to combine map layers are formalized with algebraic specifications and the optimization attempts to identify a more efficient combination of the initial operations. It employs algebraic methods to substitute complex combinations by simpler ones (Ullman 1982), a technique commonly employed in electrical engineering applications (Preparata and Yeh 1973) and compiler design (Aho *et al.* 1985).

The remainder of this paper is structured as follows: the next section briefly introduces the two formalisms used, i.e., algebraic specifications and decision tables. Section 3 formalizes a specific map overlay operation and then generalizes this formalism so that it becomes applicable for arbitrary overlay operations. The properties of value operations to combine layers, cell-by-cell, are analyzed in Section 4, and strategies are proposed for efficient combinations of multiple overlay operations. Section 5 summarizes the results and concludes with directions for further research.

2 Formal Methods

2.1 Algebras

Multi-sorted algebraic specifications are a tool commonly used in software engineering to describe completely and formally the behavior of complex systems (Liskov 1986). They are based on heterogeneous algebras (Birkhoff and Lipson 1970) and their extensions to multi-sorted algebras (Guttag 1977). Data algebras (Zilles 1979), using equations to define the independent properties of data structures, and abstract data types (ADTs) (Goguen *et al.* 1978) influenced today's understanding of algebraic specifications. An *algebraic specification* consists of three parts (Liskov 1986, Ehrich *et al.* 1989): (1) a set of *sorts*,[1] (2) a set of *operations* defined upon the sorts, and (3) a set of *axioms* or *equations* that specifies the behavior of the operations. Two kinds of operations are distinguished: (1) operations to create or modify an ADT, called *creators*, and (2) operations to observe some of the properties of an ADT, called *observers*. A specification is sufficiently correct and sufficiently complete in terms of its creators and observers (Guttag and Horning 1978).

The following example specifies an ADT *point* in the Euclidean plane. The ADTs *integer* and *boolean* are assumed to exist with their usual semantics. The syntax of this specification method resembles the syntax of specification-like programming languages such as Eiffel (Meyer 1988) and MOOSE (Egenhofer and Frank 1988).

[1]The term *sort* does not imply an order (sorting) over the instances. Programming languages use the less ambiguous term *type* in lieu of *sort*, however, *types* consider also the structure of the *sorts* (Cardelli and Wegner 1985) which is part of an implementation.

```
SORTS[2]       point USES integer, boolean
OPERATIONS[3]  make:  integer × integer → point
               x:  point → integer
               y:  point → integer
               isEqual:  point × point → boolean
VARIABLES[4]   i1, i2:  integer; p1, p2:  point
EQUATIONS[5]   x (make (i1, i2)) == i1
               y (make (i1, i2)) == i2
               isEqual (p1, p2) == integer.isEqual (x (p1), x (p2)) and
                                   integer.isEqual (y (p1), y (p2))
```

Specification 1: Point.

2.2 Decision Tables

A *decision table* is another method to specify formally the behavior of operations, particularly those which can be described by a series of rules. It consists of two parts: (1) a set of *conditions* which have to be satisfied simultaneously and (2) the corresponding *actions* to be taken upon the conditions (Metzner and Barnes 1977).

Decision tables are most naturally presented in the form of a table with the set of conditions being put into the upper half of the table and the set of corresponding actions underneath. Boolean values, T and F, are assigned to the conditions indicating whether or not the corresponding action should be taken. If an action is taken independent of a condition then a dash in the corresponding decision indicates *don't care.* Since the entries in conditions and corresponding actions are logically connected with *AND*, they are commutative.

Decision tables are a well-suited tool to express some spatial analysis operations which frequently use complex algebraic expressions to describe their operations and mappings. The following example demonstrates the use of a decision table to formalize a particular value operation, the *localRating*, frequently used in the MAP algebra (Tomlin 1990). LocalRating assigns to each n-tuple of values a new value. For example, the following localRating combines a layer of *altitudes* with a *vegetation* layer into a new layer *windExposure.*

- If the altitude is greater than or equal to 290 and vegetation type 0 then the wind exposure is 1.
- If the altitude is greater than or equal to 290 and vegetation types 1–3 then the wind exposure is 2.
- If the altitude is less than 290 and vegetation type 0, 1, or 3 then the wind exposure is 3.

[2]The SORTS definition includes the data type to be specified and the types it USES to describe its properties.

[3]OPERATIONS are defined by their name, the Cartesian product of the input sorts, and the sort of the result.

[4]VARIABLES describe the instances of the sorts used in the equations.

[5]The behavior of each operation is expressed by EQUATIONS in terms of equivalent observe and create operations.

- If the altitude is less than 290 and vegetation type 2 then the wind exposure is 4.

Table 1 shows a decision table which models these rules.

altitude vegetation	≥ 290 0	≥ 290 $1 \vee 2 \vee 3$	< 290 $0 \vee 1 \vee 3$	< 290 2
windExposure	1	2	3	4

Table 1: The decision table for the local rating of altitudes and vegetation [Tomlin 1989].

3 Formalizing Overlay Operations

The raster model is a particular subclass of the regular tessellations with a discrete representation of space (Egenhofer and Herring 1991, Frank 1991b). It partitions the area of interest into equally-shaped *cells* so that (1) the set of all cells forms a complete partition, called a *layer*, and (2) any pair of cells does not overlap. This section will demonstrate the use of algebraic specifications to specify formally combinations of layers.

3.1 An Overlay Example

The most common queries upon layers are based on the *map overlay* methodology, i.e., the combination of several layers into a new one (Steinitz *et al.* 1976). A simple, but specific example is to show the use of algebraic specifications for describing a particular overlay operation. The operation to be specified is the combination of the two layers with regular rectangular cells, both over the same spatial extent, in the same scale, and with the same orientation. Each cell is made from a *location* and a *value*. In this particular example, each value is an integer, with the operations equal and maximum and their usual semantics, and each location is a rectangle described by its lower-left and upper-right points (Specification 1), a creator (make), and three observe operations (Specification 2).

```
SORTS       location USES point, boolean
OPERATIONS make:  point × point → location
            lowerLeft:  location → point
            upperRight:  location → point
            isEqual:  location × location → boolean
VARIABLES   p₁, p₂:  point, l₁, l₂:  location
EQUATIONS   lowerLeft (make (p₁, p₂)) == p₁
            upperRight (make (p₁, p₂)) == p₂
            isEqual (l₁, l₂) == point.isEqual (lowerLeft (l₁),
                                               lowerLeft (l₂)) and
                                point.isEqual (upperRight (l₁),
                                               upperRight (l₂))
```

Specification 2: Location as the Cartesian product of two points.

Cells have operations to *make* a new one and to access its components, i.e., *getLocation* and *getValue* (Specification 3).

```
SORTS       cell USES location, value
OPERATIONS make:  location × value → cell
            getLocation:  cell → location
            getValue:  cell → value
VARIABLES   l:  location; v:  value
EQUATIONS   getValue (make (l, v)) == v
            getLocation (make (l, v)) == l
```

Specification 3: Cells.

The resulting layer contains the greater of the two values at the corresponding spatial locations (Specification 4).

```
SORTS       layer USES cell
OPERATIONS make:  cell × cell × ... × cell → layer
            overlayMaximum:  layer × layer → layer
VARIABLES   l₁, l₂:  layer; c₁, c₂, c₃:  cell
EQUATIONS   FOR EACH [c₁, c₂, c₃] IN [l₁, l₂, overlayMaximum (l₁, l₂)]:
                location.isEqual (cell.getLocation (c₃),
                                  cell.getLocation (c₁)) and
                location.isEqual (cell.getLocation (c₃),
                                  cell.getLocation (c₂)) and
                value.isEqual (cell.getValue (c₃),
                value.maximum (cell.getValue (c₁), cell.getValue (c₂))).
```

Specification 4: Combining two layers by selecting the maximum value.

The syntax of the equations uses a FOR EACH[5] loop (Liskov *et al.* 1981) to apply an operation to all elements of a set (Backus 1978), i.e., all cells which are part of, or IN, a layer. Actually, this is an observe operation upon a layer returning the cells in the aggregate one after another. Simultaneous loops over multiple aggregates group the parts and the aggregates pairwise between brackets so that the n-th part in on bracket corresponds with the n-th aggregate in the other.

This set of specifications for layers, cells, rectangles, and integers completely formalizes the behavior of this particular overlay operation.

- Layers are combined by applying a particular operation to corresponding cells, i.e., cells with the same spatial location.
- The same value operation is applied to all cells of a layer.
- The value combination of cells preserves the locations of the cells, i.e., the location of each cell in the resulting layer is the same as the one of the cells combined.

3.2 A Generalized Overlay Operation

The previous specification can be generalized so that it holds for other overlay operations as well. Such a generic specification is based upon the definition of a generalized value type, a superclass of all possible sorts which may characterize the non-spatial properties of a cell.

A value type must provide operations to compare two values for equality (`isEqual`) and to `combine` values (Specification 5). The specification of its `create` operation is `DEFERRED` (Meyer 1988), because it depends upon the particular value type used.

```
SORTS       value USES boolean
OPERATIONS  create:   DEFERRED → value
            isEqual:   value × value → boolean
            combine:   value × value ×...× value → value
```

Specification 5: A generic value.

Likewise, the location specification may vary for different shapes of cells. Besides the `make` operation, the ADT location must provide an operation to compare two locations for equivalence (`isEqual`) (Specification 6).

```
SORTS       location USES boolean
OPERATIONS  make:  DEFERRED → location
            isEqual:  location × location → boolean
```

Specification 6: A generic location.

[5]Not to be confused with the for-all quantifier, ∀, commonly used in calculus.

The specification of the ADT cell as the Cartesian product of location and value stays unchanged (Specification 3). The modified ADT layer has a single overlay operation with varying implementations depending on the value operation used to combine corresponding cells. The FOR EACH loop runs over the sets of all cells in all layers, indicated by c_i and l_i, respectively (Specification 7).

```
SORTS       layer USES cell
OPERATIONS  make:  cell × cell ×...× cell → layer
            overlay:  layer × layer ×...× layer × value.combine → layer
VARIABLES   c_i, c_n:  cell; l_i:  layer
EQUATIONS   FOR EACH [c_i, c_n] IN [l_i, overlay (l_i, value.combine)]:
                 location.isEqual (cell.getLocation (c_n),
                                   cell.getLocation (c_i)) and
                 value.isEqual (cell.getValue (c_n),
                                value.combine (cell.getValue (c_i)))
```

Specification 7: A parametric layer.

The behavior of any overlay operation is expressed by a particular operation upon the values of individual cells (`value.combine`). The usage of a variable argument over value operations reduces the specification to a single, generic operation. `Combine` is similar to the operators *apply* (Scholl and Voisard 1989) and λ (Güting 1988) in other formalizations.

The generalized overlay specification reveals that the characteristics of these overlay operations are exclusively determined by the operation combining several values. Conversely, the properties of the value operation immediately map onto the properties of the overlay operation. For arithmetic overlay operations, it is sufficient to consider each layer as a set of cells, i.e., no topological relationships among the cells are used. Since the values are combined over the same location, the overlay operation—in terms of relational algebra (extended with arithmetic capabilities) (Ullman 1982)—is (1) an equijoin over the same location (Frank 1987) followed by (2) an arithmetic operation `combining` the values of corresponding location and (3) a projection of the locations and the combined value.

4 Optimization

An overlay operation over multiple layers results in a new layer which, in turn, may be used as an argument in another overlay operation. Frequently, many overlay operations are combined this way to perform a more complex operation (Tomlin 1990). While sophisticated spatial data structures may efficiently implement an individual overlay operation, they generally provide only little support for improving the processing of a series of overlays. It will quickly become time consuming to process sequentially each overlay operation by producing an intermediate layer after each operation. In lieu of immediately performing each operation, it is more efficient to evaluate first the entire operation and identify an execution strategy which predicts the shortest processing time. Similar considerations within the relational algebra to gain better performance for complex, combined operations led to the area of *query optimization* (Ullman 1982). To date, only few attempts have been made to improve systematically spatial query processing (Hudson 1989, Ooi and Sacks-Davis 1989). Current overlay processors calculate interactively one overlay at a time (Pazner *et al.* 1989), though there have been recently attempts to pursue more efficient processing strategies (Yost and Skelton 1990). To improve the overlay operations of several layers, two strategies are investigated: (1) to identify equivalent sub-expressions so that they can be computed only once, and (2) to integrate several individual overlay operations into a single one. Both strategies will be investigated subsequently.

4.1 Notation

The uppercase Greek letter *omega* (Ω) will be used to denote overlay. Its arguments are (1) the ordered set of layers $layer_1, \ldots, layer_n$ with $n > 0$, and (2) a particular *combination* operation (Equation 1).

$$\Omega_{combination}(layer_1, \ldots, layer_n) \tag{1}$$

The combination operation may be a function, such as *max* or *average*, or a decision table. For instance, the value combination specified in decision table 1 is applied to the layers altitudes and vegetation, resulting in the layer windExposure (Equation 2).

$$windExposure := \Omega_{Table\ 1}\ (altitude, vegetation) \tag{2}$$

4.2 Equivalent Overlay Operations

A first step during processing the combination of overlays is to identify those sequences of operations that occur several times so that they need to be executed only once. The goal for such an overlay optimizer is to find equivalent, but more efficient expressions, i.e., expressions which yield the same result within less time. This strategy requires a formal knowledge of equivalent expressions. Mathematics has the notion of properties of combinations of operations to describe whether two expressions are equivalent or not. Most familiar are the commutative, associative, and distributive laws, e.g., for the combinations of sets with the operations union and intersection. Likewise, the combination of layers with various overlay operations may be described by their commutative (Equation 3), associative (Equation 4), and distributive (Equation 5) properties.

$$\Omega_{combination}(layer_1, layer_2) = \Omega_{combination}(layer_2, layer_1) \tag{3}$$

$$\begin{aligned}&\Omega_{combination}(\Omega_{combination}(layer_1, layer_2), layer_3) = \\ &\quad \Omega_{combination}(layer_1, \Omega_{combination}(layer_2, layer_3))\end{aligned} \tag{4}$$

$$\begin{aligned}&\Omega_{combination_1}(layer_1, \Omega_{combination_2}(layer_2, layer_3)) = \\ &\quad \Omega_{combination_2}(\Omega_{combination_1}(layer_1, layer_2), \Omega_{combination_1}(layer_1, layer_3))\end{aligned} \tag{5}$$

The specification of the generalized overlay operation (Specification 7) demonstrated that an overlay varies only over different value operations; therefore, the properties of the combinations of overlay operations can be based upon the properties of the corresponding value operation. For instance, the combination of three layers is associative if and only if the value operation is associative as well:

$$\Omega_{combination}(\Omega_{combination}(layer_1, layer_2), layer_3) = \\ \Omega_{combination}(layer_1, \Omega_{combination}(layer_2, layer_3)) \tag{6}$$

$$\Leftrightarrow$$

$$value.combine(value.combine(v_1, v_2), v_3) = \\ value.combine(v_1, value.combine(v_2, v_3)) \tag{7}$$

Since the overlay operations depend completely upon the corresponding value operations, they can be optimized by only considering the value operations in the same sequence as the corresponding overlay operations. Equivalent overlay operations can be found by analyzing the properties of the value operations. These properties are described in the axioms of the specifications of the values. For example, given a complex query containing the following expressions:

$$\ldots \Omega_{add}(layer_1, \Omega_{add}(layer_2, layer_3)) \ldots \Omega_{add}(layer_3, \Omega_{add}(layer_2, layer_1)) \ldots \tag{8}$$

The axioms of the particular value specifications may provide the necessary information about the properties of the add operation, e.g.,

```
SORTS       value
OPERATIONS  add:  value × value → value
VARIABLES   v1, v2, v3:  value
EQUATIONS   add (v1, v2) == add (v2, v1)
            add (v1, (add (v2, v3)) == add (add (v1, v2), v3))
```

Specification 8: Commutative and associative properties of the value operation *add.*

Based upon these axioms it can be formally analyzed whether or not these two expressions are the same. First, the overlay operation is substituted by the corresponding operations upon values (Equations 9 and 10).

$$\Omega_{add}(layer_1, \Omega_{add}(layer_2, layer_3)) \Rightarrow value.add(v_1, value.add(v_2, v_3)) \tag{9}$$

$$\Omega_{add}(layer_3, \Omega_{add}(layer_2, layer_1)) \Rightarrow value.add(v_3, value.add(v_2, v_1)) \tag{10}$$

Then the axioms are applied. With the associative law, Equation (10) is transformed.

$$value.add(v_3, value.add(v_2, v_1)) = value.add(value.add(v_3, v_2), v_1) \tag{11}$$

Finally, the commutative law is applied twice.

$$\begin{aligned} value.add(value.add(v_3, v_2), v_1) &= value.add(v_1, value.add(v_3, v_2)) \\ &= value.add(v_1, value.add(v_2, v_3)) \end{aligned} \tag{12}$$

Equation (12), the equivalent for (10), is the same as (9), i.e., the two subexpressions in Equation (8) are the same and, therefore, only one of them must be executed.

4.3 Integration of Multiple Overlay Combinations

A second strategy to reduce the execution time of a complex combination of overlays is to integrate several overlay operations into a single, equivalent one, i.e.,

$$\Omega_{combination_1}(layer_1, \Omega_{combination_2}(layer_2, layer_3)) \Rightarrow \Omega_{combination_3}(layer_1, layer_2, layer_3) \quad (13)$$

Again, the specification of the generalized overlay operation (Specification 7) was fundamental in tackling this problem. It shows that this integration means to move a value operation, `value.operation`$_2$, from the inner FOR EACH loop into the outer loop and combine `value.operation`$_1$ with `value.operation`$_2$ into `value.operation`$_3$. The validity of such combinations can be checked with the axioms specifying the value ADTs.

```
FOR EACH ($t_1$, $t_2$) IN (A,
        FOR EACH ($t_3$, $t_4$) IN (B, C)
                DO value.operation$_2$)
        DO value.operation$_1$
⇒
FOR EACH ($t_1$, $t_2$, $t_3$) IN (A, B, C)
        DO value.operation$_3$
```

An alternative approach to this symbolic optimization is the use of decision tables to evaluate the combinations. Given the sets of values on each layer, the decision tables can be applied to analyze the property of the combination of operations.

The following example demonstrates such an integration. Four layers, A_1, A_2, B_1, and B_2, with the four respective sets of values, {2, 4, 8}, {6, 10}, {3, 4}, and {1, 2, 3}, should be combined such that

$$result := \Omega_{min}(\Omega_{average}(A_1, A_2), \Omega_{Table\ 2b}\ (B_1, B_2)) \quad (14)$$

The decision table 2 shows the combinations for the two inner overlays.

A_1	2	2	4	4	8	8
A_2	6	10	6	10	6	10
X_1	4	6	5	7	7	9

B_1	3	4	4
B_2	-	$1 \vee 3$	2
X_2	2	1	3

Table 2: (a) $X_1 := average(A_1, A_2)$ and (b) $X_2 := (B_1, B_2)$.

Table 3 shows the result of the combination of the two intermediate results with the operation *min*.

X_1	4	4	4	6	6	6	5	5	5	7	7	7	9	9	9
X_2	2	1	3	2	1	3	2	1	3	2	1	3	2	1	3
X	2	1	3	2	1	3	2	1	3	2	1	3	2	1	3

Table 3: $X := min(X_1, X_2)$.

The sequence of operations may be expressed by a single table. Its condition part contains the Cartesian product of the values in the four layers and the action part has the corresponding values of the combinations (Table 4).

A_1	2	2	2	4	4	4	2	2	2	4	4	4	8	8	8	8	8	8
A_2	6	6	6	6	6	6	10	10	10	10	10	10	6	6	6	10	10	10
B_1	4	3	4	4	3	4	4	3	4	4	3	4	4	3	4	4	3	4
B_2	$1 \vee 3$	–	2	$1 \vee 3$	–	2	$1 \vee 3$	–	2	$1 \vee 3$	–	2	$1 \vee 3$	–	2	$1 \vee 3$	–	2
X	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3

Table 4: $X := min(average(A_1, A_2), Table\ 2b(B_1, B_2))$.

Table 4 can be simplified by combining columns with the same actions, e.g.,

A_1	$2 \vee 4 \vee 8$	$2 \vee 4 \vee 8$	$2 \vee 4 \vee 8$	$2 \vee 4 \vee 8$	$2 \vee 4 \vee 8$	$2 \vee 4 \vee 8$
A_2	6	10	6	10	6	10
B_1	4	4	3	3	4	4
B_2	$1 \vee 3$	$1 \vee 3$	–	–	2	2
X	1	1	2	2	3	3

Table 5: $X := min(average(A_1, A_2), Table\ 2b(B_1, B_2))$.

Further integrations (over the values of A_2) and the substitutions of disjunctions which cover the entire domain by the value *don't care* reduce the value operation to an operation which is independent of the two layers A_1 and A_2 (Table 6); therefore, the entire overlay operation may be reduced to the combination of the two layers B_1 and B_2.

A_1	–	–	–
A_2	–	–	–
B_1	4	3	4
B_2	$1 \vee 3$	–	2
X	1	2	3

Table 6: The simplified decision table for $\Omega_{min}(\Omega_{average}(A_1, A_2), \Omega_{Table\ 2b}(B_1, B_2))$.

The decision table also indicates in which order the two layers should be processed. The value of a layer needs not be examined if the result is independent of it. For example, it is more efficient to execute $\Omega_{Table\ 2b}(B_1, B_2)$ than $\Omega_{Table\ 2b}(B_2, B_1)$. In the first case, the result of half of the operations is determined by just examining B_1, because the outcome of the combination with value 3 is independent of the value at the corresponding location in B_2. If the value is 4 then the corresponding value in B_2 must be examined as well. On the other hand, if the converse operation is executed then always the values of both layers must be processed.

5 Conclusion

Rigid formal methods have shown to be effective tools to identify optimization strategies for combinations of overlay operations. The algebraic specification of a generalized overlay operation for tessellations revealed that

- a layer may be considered a set of cells, each consisting of a location and a value, and
- arithmetic overlay operations over layers can be broken down into a value operation to be performed for each cell of a layer or tuple of corresponding cells in several layers, similar to the application of a function to a whole set in functional programming.

Since overlay operations are founded upon value operations, it is possible to map the considerations about best execution plans for operations onto considerations about the combination of value operations. Two particular ways of optimizing several overlay operations have been investigated:

1. the use of axiomatic description of the value operations to identify whether or not two combinations of value operations are equivalent. Faster executions of combinations of overlays are possible, because such equivalent subexpressions can be substituted by the result of one single overlay operation.
2. the use of decision tables, representing the characteristics of value operations, to integrate several overlay operations. This method can be applied if the sets of values of all layers are known. The integration reduces any combination of overlay operations into a single one and is most effective if the number of conditions is small. Decision tables are less suitable for large sets of conditions, because the tables grow multiplicatively before reduction.

The results obtained demonstrated the usefulness of the approach. Further investigations are necessary to build sophisticated query optimizers for raster GIS's. The present work, intentionally, excluded geometric operations on cells, e.g., those which exploit the neighborhood relationship between cells. Within the formal framework provided it is now possible to study their behavior to formalize geometric operations on tessellations.

6 Acknowledgements

Thanks to Andrew Frank for his useful comments on earlier versions of this paper and to Kelly Chan, Werner Kuhn, Hans-Peter Kriegel, and Alan Saalfeld for stimulating discussions.

References

A. Aho, R. Sethi, and J. Ullman. Compilers: Principles, Techniques, and Tools. Addison-Wesley Publishing Company, Reading, MA, 1985.

J. Backus. Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs. Communications of the ACM, 21(8):613–641, August 1978.

G. Birkhoff and J. Lipson. Heterogeneous Algebras. Journal of Combinatorial Theory, 8:115–133, 1970.

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

K. Chan and D. White. Map Algebra: An Object-Oriented Implementation. In: International Geographic Information Systems (IGIS) Symposium: The Research Agenda, Vol. II, pages 127–150, Arlington, VA, November 1987.

S.K. Chang, E. Jungert, and Y. Li. The Design of Pictorial Databases Based Upon the Theory of Symbolic Projections. In: A. Buchmann, O. Günther, T. Smith, and Y. Wang, editors, Symposium on the Design and Implementation of Large Spatial Databases, Lecture Notes in Computer Science, Vol. 409, pages 303–323, Springer-Verlag, New York, NY, July 1989.

E.F. Codd. A Relational Model for Large Shared Data Banks. Communications of the ACM, 13(6):377–387, June 1970.

J. Corbett. Topological Principles of Cartography. Technical Report 48, Bureau of the Census, Department of Commerce, 1979.

M. Egenhofer and A. Frank. MOOSE: Combining Software Engineering and Database Managemenst Systems. In: Second International Workshop on Computer-Aided Software Engineering, Advance Papers, Cambridge, MA, May 1988.

M. Egenhofer, A. Frank, and J. Jackson. A Topological Data Model for Spatial Databases. In: A. Buchmann, O. Günther, T. Smith, and Y. Wang, editors, Symposium on the Design and Implementation of Large Spatial Databases, Lecture Notes in Computer Science, Vol. 409, pages 271–286, Springer-Verlag, New York, NY, July 1989.

M. Egenhofer and J. Herring. A Mathematical Framework for the Definition of Topological Relationships. In: K. Brassel and H. Kishimoto, editors, Fourth International Symposium on Spatial Data Handling, pages 814–819, Zurich, Switzerland, July 1990.

M. Egenhofer and J. Herring. High-Level Spatial Data Structure. in: D. Maguire, D. Rhind, and M. Goodchild, editors, Geographical Information Systems: Overview, Principles, and Applications, Longman Scientific and Technical, London, 1991 (in press).

H.-D. Ehrich, M. Gogolla, and U. Lipeck. Algebraic Specifications of Abstract Data Types (in German). B.G. Teubner, Stuttgart, 1989.

A. Frank and W. Kuhn. Cell Graph: A Provable Correct Method for the Storage of Geometry. In: D. Marble, editor, Second International Symposium on Spatial Data Handling, pages 411–436, Seattle, WA, 1986.

A. Frank. Overlay Processing in Spatial Informaion Systems. In: N. Chrisman, editor, AUTO-CARTO 8, Eighth International Symposium on Computer-Assisted Cartography, pages 16–31, Baltimore, MD, March 1987.

A. Frank. Qualitative Spatial Reasoning about Cardinal Directions. In: D. Mark and D. White, editors, Autocarto 10, Baltimore, MD, March 1991a.

A. Frank. Spatial Concepts, Geometric Data Models and Data Structures. Computers and Geo-Sciences, 1991b (in press).

A. Frank and D. Mark. Language Issues for Geographical Information Systems. in: D. Maguire, D. Rhind, and M. Goodchild, editors, Geographical Information Systems: Overview, Principles, and Applications, Longman Scientific and Technical, London, 1991 (in press).

J. Goguen, J. Thatcher, and E. Wagner. An Initial Algebra Approach to the Specification, Correctness, and Implementation of Abstract Data Types. In: R. Yeh, editor, Current Trends in Programming Methodology, Prentice-Hall, Englewood Cliffs, NJ, 1978.

J. Goguen. Modular Algebraic Specification of Some Basic Geometrical Constructions. In: D. Kapur and J. Mundy, editors, Geometric Reasoning, pages 123–153, MIT Press, Cambridge, MA, 1989.

M. Goodchild. A Geographical Perspective on Spatial Data Models. Computers and Geo-Sciences, 1991 (in press).

R. Güting. Geo-Relational Algebra: A Model and Query Language for Geometric Database Systems. In: J. Schmidt, S. Ceri, and M. Missikoff, editors, Advances in Database Technology—EDBT '88, International Conference on Extending Database Technology, Venice, Italy, Lecture Notes in Computer Science, Vol. 303, pages 506–527, Springer Verlag, New York, NY, 1988.

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

J. Guttag and J. Horning. The Algebraic Specification of Abstract Data Types. Acta Informatica, 10:27–52, 1978.

J. Herring. TIGRIS: Topologically Integrated Geographic Information Systems. In: N. Chrisman, editor, AUTO-CARTO 8, Eighth International Symposium on Computer-Assisted Cartography, pages 282–291, Baltimore, MD, March 1987.

D. Hudson. A Unifying Database Formalism. In: ASPRS/ACSM Annual Convention, pages 146–153, Baltimore, MD, April 1989.

B. Liskov, R. Atkinson, T. Bloom, E. Moss, J.C. Schaffert, R. Scheifler, A. Snyder. CLU Reference Manual. Lecture Notes in Computer Science, Vol. 114, Springer-Verlag, New York, NY, 1981.

B. Liskov and J. Guttag. Abstraction and Specification in Program Development. MIT Press, Cambridge, MA, 1986.

W. Mallgren. Formal Specification of Graphic Data Types. ACM Transactions of Programming Languages and Systems, 4(4):687–710, October 1982.

J. Metzner and B. Barnes. Decision Table Languages and Systems. Academic Press, New York, NY, 1977.

B. Meyer. Object-Oriented Software Construction. Prentice Hall, New York, NY, 1988.

G. Nagy and S. Wagle. Geographic Data Processing. ACM Computing Surveys, 11(2):139–181, June 1979.

B. Ooi and R. Sacks-Davis. Query Optimization in an Extended DBMS. In: W. Litwin and H.-J. Schek, editors, Third International Conference on Foundations of Data Organization and Algorithms (FODO), Lecture Notes in Computer Science, Vol. 367, pages 48–63, Springer-Verlag, New York, NY, June 1989.

M. Pazner, K.C. Kirby, and N. Thies. MAP II: Map Processor—A Geographic Information System for the Macintosh. John Wiley & Sons, New York, NY, 1989.

T. Peucker and N. Chrisman. Cartographic Data Structures. The American Cartographer, 2(2):55–69, 1975.

D. Peuquet. A Conceptual Framework and Comparison of Spatial Data Models. Cartographica, 21(4):66–113, 1984.

D.J. Peuquet and Z. Ci-Xiang. An Algorithm to Determine the Directional Relationship Between Arbitrarily-Shaped Polygons in the Plane. Pattern Recognition, 20(1):65–74, 1987.

F. Preparata and R. Yeh. Introduction to Discrete Structures. Addison-Wesley Publishing Company, Reading, MA, 1973.

H. Samet. The Design and Analysis of Spatial Data Structures. Addison-Wesley Publishing Company, Reading, MA, 1989.

M. Scholl and A. Voisard. Thematic Map Modeling. In: A. Buchmann, O. Günther, T. Smith, and Y. Wang, editors, Symposium on the Design and Implementation of Large Spatial Databases, Santa Barbara, CA, Lecture Notes in Computer Science, Vol. 409, pages 167–190, Springer-Verlag, New York, NY, July 1989.

T. Smith and A. Frank, Report on Workshop on Very Large Spatial Databases. Journal of Visual Languages and Computing, 1(3):291–309, 1990.

C. Steinitz, P. Parker, and L. Jorden. Hand-Drawn Overlays: Their History and Prospective Uses. Landscape Architecture, 66(8):444–455, 1976.

A. Tarski. What is Elementary Geometry? in: L. Henkin, P. Suppes, and A. Tarski, editors, Symposium on the Axiomatic Method, pages 16–29. North Holland, Amsterdam, 1959.

C.D. Tomlin. Digital Cartographic Modeling Techniques in Environmental Planning. PhD thesis, Yale University, 1983.

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

J. Ullman. Principles of Database Systems. Computer Science Press, Rockville, MD, 1982.

M. White. Technical Requirements and Standards for a Multipurpose Geographic Data System. The American Cartographer, 11(1):15–26, March 1984.

M. Yost and B. Skelton. Programming Language Technology for Raster GIS Modeling. In: GIS/LIS 90, pages 319–327, Anaheim, CA, November 1990.

S. Zilles. An Introduction to Data Algebras. In: D. Bjørner, editor, Abstract Software Specifications, pages 248–272, Spring-Verlag, New York, 1979.