Many Sorted Algebraic Data Models for GIS

FENG-TYAN LIN

Graduate Institute of Building and Planning, National Taiwan University,

#1, Sec. 4, Roosevelt Road, Taipei, Taiwan.

TEL: (886)-2-23638711 ext.37

FAX: (886)-2-23659197

e-mail: ftlin@ccms.ntu.edu.tw

Abstract. Although many GIS data models are available, a declarative, operational, well-defined, implementation-independent, and object-oriented language is lacking. Based on the theory of many sorted algebra, this work presents a family of geometric data models. Some geographical data models of urban information systems are illustrated using homomorphism. According to the results, the preferred characteristics of mixing declarative and operational statements, multiple representations, tight interdependency among objects, and integration of vector and raster based systems can be achieved through this mechanism.

1. Introduction

Data models, as abstractions of the real world, capture and organize certain features, behaviors and operators that are relevant to particular applications in different ways. Moreover, to yield a better performance and analysis power, different data models may correspond to different data structures and manipulation functions (Maguire and Dangermond 1991, Peuquet 1984, Goodchild 1991).

In Geographical Information Systems (GIS), many data models have been discussed and developed for quite some time. Among them, raster and vector data models are the conventionally used ones. A typical raster data model represents geographical information as a collection of layers or themes, which are subsequently composed of cells in some particular order. On the other hand, a typical vector data model describes geographical features as points, lines or polygons by coordinates. However, they are not necessarily two independent data models. Some efforts concerning their combination are currently underway (Piwowar *et al.* 1990, Chen 1997). To gain more capabilities, this work suggests many sorted algebra as a high level language of GIS data models.

The concept of many sorted algebra, originating from the theory of abstract algebra in modern mathematics, aptly serves as a novel data model language (Goguen *et al.* 1976, Futasugi *et al.* 1985, Jaffar *et al.* 1986). The many sorted algebra has the features of being object-oriented, declarative, operational, well defined, and implementation-independent (Breu 1991). However, this work will heavily stress the end users' perspectives, and concern itself primarily with the validity of the expected systems instead of the systems' efficiency, which is the responsibility of system developers and programmers. Therefore, only issues of correctness and not efficiency will be discussed.

The rest of this paper is organized as follows. Section 2 refers to the previous work carried out in this area and their relevance to this work. Section 3 reviews the hierarchy of data models and describes the relations among algebraic models. Section 4 cites urban information systems (UIS) as an example to illustrate some of the features of a complex GIS. Section 5 highlights the basic concepts of many sorted algebra in mathematics. Thereafter, some primitive geometric algebras are constructed. Additionally, a description is made of how to use the many sorted algebra to integrate and realize first order logic, object-oriented mechanisms, operational and declarative statements. Next, Section 6 introduces the family of geometric algebras in more detail. This section also discusses canonical forms, entity construction mechanisms and operational functions. Section 7 demonstrates how the technique of homomorphism between geometric and geographic algebras can be employed to handle some of preferred features in data models by examples in complex urban information systems. Concluding remarks and recommendations for future studies are made in Section 8. Some geometric algebras are shown in the appendix.

2. Background

The design of data models is one of major concerns in computer science. Before developing many sorted algebraic data models for GIS, it is worth reviewing the previous work to earn some experiences from them and get further insight into the essence and construction mechanism of data models.

Among various issues, selecting languages that specify data properties and functions is an essential decision, since data model languages can vary in form, e.g. diagrams, graphs, plain sentences, program-like statements, pseudo-codes, or strictly mathematical expressions (Ullman 1982, Laurini and Thompson 1992). Different languages express conceptual data models in varying degrees of communicability, understanding, abstraction and preciseness. Diagrams, graphs and plain sentences can express the entire concept of data models. Although data models expressed in this

way appear easily understood, they are innately imprecise and leave too many possible interpretations, potentially leading to different semantics. Consequently, such models can be ambiguous so that the users and the programmers might have conflicting interpretations.

Although program-like statements and pseudo-codes express data models more precisely (Car 1995, Kusters *et al.* 1997), they typically rely on some predefined, pre-confined, programming schemes via operational statements. However, they lack declarative capabilities in most cases. More specifically, most data models are expressed from the programmers' perspectives that, unconsciously, emphasize the feasibility of implementing the program. However, data models must also be constructed from the perspectives of end users who (a) might have no programming experience and (b) are interested in specifying the expected systems instead of stressing how to efficiently implement the systems. Moreover, it is quite possible that the systems proposed by the end users offer no available solution. Therefore, such a possibility implies that languages for data models should be able to express problems correctly as well as solve problems efficiently. This is despite the fact that these two capabilities can be used in different circumstances or system development stages.

To allow for such capabilities, the language of relational data bases has two schemes: algebra for operational statements and logic calculus for declarative statements (Date 1990). However, from a mathematical sense, algebra is not only operational but also declarative. Restated, logic calculus can be transformed and therefore integrated into algebraic expressions. In this work, we elucidate on how to precisely describe data models from both the programmers' and the end users' perspectives in a consistent manner by using mathematical algebra in this broader meaning. However, we do not deny the merits of plain texts, diagrams and graphs for communicative purposes. In addition, any combination of expressive forms should be freely employed if deemed necessary and appropriate in practice.

As is well recognized, general users conceive geographical objects, such as rivers, buildings, cities and mountains, instead of a collection of cells, points, lines, and/or polygons. This fact implies that data models should be able to describe objects more conceptually. Object-oriented data modelling reveals some interesting possibilities for expressing geographic features in a more natural manner. The programming languages most frequently cited as originating and popularizing the object-oriented programming paradigms are Simula, Smalltalk, C++, CLOS and CLU. However, these programming languages realize the concept of the object-oriented paradigm in different ways (Cattell 1991, Graham 1991). This flourishing but somewhat confused situation further implies the urgent need for a language that is more declarative, implementation-independent, and even possesses a higher degree of abstraction

(Pressman 1993). This extremely high level language would enable users to model the real world using conventional terms in their own domain applications without worrying about the implementation details. Meanwhile, the semantics would still be correctly preserved when translating from the very high level language to machine codes.

Owing to their rigid syntactic and semantic properties, algebraic languages have been shown that they have the potentiality to fulfil the requirements mentioned above, and have been used to study various abstract systems, including raster and vector based data models, for quite some time. Tomlin (1991) developed map algebra to manipulate raster based map layers. Ritter *et al.* (1990) presented image algebra to describe image processing algorithms. Takeyama and Couclelis (1997) further generalized the previous notions to establish Geo-Algebra that is aiming to provide a rigorous common formalism for raster map analysis and spatial modelling. Couclelis (1997) also demonstrated that cellular automata could be formulated in terms of Geo-Algebra, while Wagner (1997) integrated cellular automata with GIS involving map algebra. On the other hand, Guting (1988) proposed Geo-Relational Algebra to store, retrieve and manipulate vector based data for GIS, VLSI- and CAD-data bases. Stiny (1991) introduced algebras of design to shape grammars. Car and Frank (1995) employed algebraic specification to formalize conceptual models for position, nodes, edges and lists of GIS using an executable code written in functional programming language Gofer. Lin and Young (1996) proposed an algebraic data model for GIS applications. Kusters *et al.* (1997) adopted the object-oriented approach and combined textual with graphical representations to specify geo-class, point, line, region, raster, network and temporal classes for requirement engineering and software design of GIS.

Contrasting with traditional algebra, which allows one data type only, many sorted algebra can handle multiple data types and possess many preferable properties. This paper employs many sorted algebra as a language for describing GIS data models and will be elaborately discussed and further developed in the sequel.

3. Algebraic Models in Data Model Hierarchy

Comparing to other data models, algebraic ones are more abstract and conceptual. Peuquet suggests four levels of abstraction relevant to geographical databases: reality, data model, data structure and file structure (Maguire and Dangermond 1991). The level of reality is the real world to be modeled. The level of the data model is an abstraction of the real world in a conceptual manner. The level of the data structure is a logical implementation of the data model and is frequently expressed in terms of

arrays, lists, trees and other complex constructs. The level of the file structure is a physical implementation of the data structures in storage hardware.

Unsurprisingly, closely examining each level of abstractions reveals that many details can be explored. As mentioned earlier, GIS models can be classified into two categories, i.e. raster and vector data models, which are in the second level. Each data model can further correspond to several logical models in the third level. For instance, the raster data model can be encoded in an array of grid squares, quad-tree blocks, or run length methods (Laurini and Thompson 1992). Meanwhile, the vector data model can be implemented in point-line-polygon or simplicial complex with lattice relations (Kainz *et al.* 1993). On the other hand, the algebraic data model, which is of primary concern in this work, should be placed in the second level.

The algebraic language describes raster and vector data models in the same syntactic structure. The algebraic language can be used to describe a data model that can be naturally implemented in vector, raster, or combination form. Therefore, the algebraic data model is placed on the top of vector and raster models. The algebraic data model can be further divided into two sub-levels: geometric and geographic data models. The geometric data model describes objects in geometric meanings, such as points, lines, polygons and cells. The geographical data model describes objects in geographic or application meanings, such as roads, blocks, power systems and zoning maps. Both geometric and geographic data models are algebras; homomorphism relations also exist between them. Figure 1 illustrates the different levels of abstractions.

4. Prominent Issues

Before examining algebraic data models, some prominent issues in GIS must be addressed by citing Urban Information Systems (UIS) as an illustrative example. Through such an example, some essential features of the powerful data models can be more fully realized.

UIS possesses many complex features of GIS. In a local government environment, the UIS consists of many fundamental databases involving various data formats and many useful functions in diverse degrees of complexity to provide various administrative and planning services in different departments. Fundamental databases, as a minimum, include basic topographic maps, plan maps, parcel maps, administrative boundary maps, street network maps, and a large number of alpha-numeric tables recording information of population, land, transportation and public services. Furthermore, along with common input, output, editing and querying functions, several important analysis functions are identified in UIS. Among these functions include overlaying, buffering, address geocoding, polygonization, relational

matching, location, allocation, routing, flow analysis, dynamic segmentation, navigation, inventory and monitoring (Cowen and Shirley 1991).

Figure 1. The structure of abstractions

In addition to the preferred properties of the object-oriented method, many issues must be addressed to establish data models for such complex information systems. Among such issues include imprecise terms, multiple views, integrity, software independence, semantic specification, declarative and operational capabilities.

Ordinary languages in our daily life are full of imprecise and ambiguous concepts. Particularly for shortening sentences, and thus communication times, many terms are used in a 'conventional' manner so that a common understanding and background knowledge of the terms are implicitly assumed. However, the imprecise usage of terms is also a source of ambiguity. Unsurprisingly, troublesome situations arise in GIS as well.

As is well known, GIS is a booming industry, as evidenced by the more than 300 commercial GIS packages available globally. Furthermore, many forthcoming

packages are being developed and promoted. However, many terms are loosely used in different ways. For instance, many GIS packages claim that they have the capability of overlaying function. Nevertheless, various implementations exist. In Figure 2(a), there are two polygons. Polygon 1 is defined by points $p1$, $p2$, $p3$ and $p4$, while polygon 2 is $p5$, $p6$, $p7$, and $p8$. In this case, many implementations of overlay function are possible. The simplest one is a visual overlaying that still contains the original two polygons. In (b), five sub-shapes are produced. However, in (c), three sub-shapes may be found in different structures. While polygon 3, formed by $p9$, $p10$, $p11$ and $p12$, is an independent one, polygon 1 could be still formed by $p1$, $p2$, $p3$ and $p4$, or grouped by two sub-shapes which are $p1$, $p9$, $p12$, $p4$ and $p10$, $p2$, $p3$, $p11$, respectively. Similar implementation may be applied to polygon 2. Therefore, at least six ways of overlaying operations could be implemented in this case.

Figure 2. Various implementations of overlaying operation

Although only one physical environment exists, a potentially infinite number of perspectives can perceive it. For instance, zoning maps describe the future development of cities in a zonal manner, while street maps record the road systems in linear form. Nevertheless, survey monument maps emphasize the point locations of the monuments in a network structure. Moreover, the concept of a road can be either represented as straight line segments in street maps or defined by city block edges. However, these thematic maps with different major concerns and perspectives (and subsequently in various scales and data structures) should be tightly integrated. The world that they represent in different ways should not have any inconsistencies, incompatibilities or contradictions. Whenever the geometric design of city blocks is changed, the road system on the street map must be simultaneously updated. This strong relationship has to be clearly addressed.

Due to limited staff, budget, technique and time, many GIS projects involving government are developed by outside consultant firms. In some administrative systems, the public departments are not allowed to explicitly specify any particular

(GIS) product in their requests for proposals. Thus, only expected functions can be included in the documents. Adopting this kind of budget and purchase system ensures that (a) the equipment and systems involving government are not monopolized by any particular service provider, and (b) each system developer has an equal opportunity to provide the best service to the government. Therefore, the specification of application systems should not be expressed in terms of any particular GIS software.

Moreover, it is also quite possible that the expected functions are unavailable in terms of current GIS functions when the requests for proposals are issued. Thus, the specification should be able to be expressed in both operational and declarative means. However, operational and declarative statements differ in terms of appearances. The order of operational statements is critical. Any change in the statement order might yield totally different results and, thus, semantics. In contrast, the order of statements does not influence the semantics of declarative sentences. Correspondingly, operational and declarative statements should be dissolved in a coherently syntactic and semantic structure.

5. Many Sorted Algebra

5.1. Mathematical basics

Since algebra preserves rigid syntactic and semantic structures in mathematical forms, some computer scientists have extended it to many sorted algebra as their theoretical foundation of programming languages (Goguen *et al.* 1976, Futasugi *et al.* 1985, Jaffar *et al.* 1986), abstract data types, object-oriented methods (Breu 1991), system development, and others. In this sub-section, basic concepts and properties of many sorted algebra are briefly introduced.

A typical algebraic system, also called *algebra* for abbreviation, consists of three components: *carrier* $\mathbf{K}$ (a family of sets of elements), *signature* Σ_K (a family of function symbols associated with the carrier $\mathbf{K}$), and a family of axioms $\mathbf{A}$. Therefore, an algebra can be denoted as a tuple $\langle \mathbf{K}, \Sigma_K, \mathbf{A} \rangle$. For simplicity, we might use the names of the carriers or the major function names in the signatures to stand for the algebras and simply denote Σ for Σ_K if this will not cause any confusion. For example, the algebra N of natural numbers has a simple structure $\langle N, \Sigma, \mathbf{A} \rangle$, where the carrier N only contains the set of natural numbers $\{0, 1, 2, \dots\}$, the signature Σ could be the set of addition and multiplication functions $\{+, \times\}$, and the set of axioms $\mathbf{A}$ conserves the reflective, associative, distributive and commutative relations. In addition to the natural number system, many well-known systems, such as the real number system *REAL* and propositional logic *BOOL*, are also shown as algebraic systems (Sahni 1985).

A conventional algebraic system consists of only one sort (data type), e.g. the real numbers, in the set of elements. However, this is not the case for the systems in the world of computer programming languages. Almost every program simultaneously uses data types of integers, real numbers, characters, and other user-defined data types through various constructs, such as array and record (Ghezzi and Jazayeri 1982). To accommodate this circumstance, the theory of many sorted algebra is developed by extending the single sorted algebra. A *many sorted Σ -algebra* is a tuple $\langle K, \Sigma, A \rangle$, where K is the *carrier* of the algebra, Σ is a set of operators (functions) $\{f_1, f_2, \dots\}$ called *signature*, and A is a set of *mandatory properties* that the algebra should follow. The carrier K is a set of sorts K_i . Each operator, say f , is a total function with domain $K_i \times K_j \times \dots \times K_n$ and range K_r , and denoted in a form of $f: K_i \times K_j \times \dots \times K_n \rightarrow K_r$. Thus, the carrier K is closed under each of the operators f_i in Σ (Sahni 1985, Hennessy 1988, Lin 1989). Mandatory properties could be axioms, constraints, equivalent equations, and logical clauses.

For every signature Σ there is a particularly important Σ -algebra called the *term algebra* for Σ . Term algebra is purely a formal one. Its carrier consists of sequences of symbols or strings, called *terms*, which are constructed using the function symbols in Σ . In other words, the carrier of the term algebra with respect to the signature Σ is a set of terms T_Σ which satisfies

- if $f \in \Sigma$ has arity 0 then the symbol f is in T_Σ .
- if $f \in \Sigma$ has arity $n > 0$ then the string of the form $f(t_1, \dots, t_n)$ is in T_Σ , whenever $t_1, \dots, t_n$ are strings in T_Σ .

Comparing the relation between Σ -algebra and its term algebra, one will find that term algebras are syntactic in nature and Σ -algebra semantics. In other words, every term could be viewed as a possible operational procedure in a sequence of functions and its corresponding meaning is the result after evaluation. This mechanism matches the common sense that different function expressions could be mapped to the same value. As a matter of fact, it has been proven that, for every Σ -algebra $\langle K, \Sigma_K \rangle$, there exists a unique Σ -homomorphism which maps syntactic terms onto elements of the carrier (Hennessy 1988).

The concept of homomorphism is very important to preserve the same properties between algebras. Let $A = \langle K, op_1, \dots, op_n \rangle$ and $A' = \langle K', op'_1, \dots, op'_n \rangle$ be two algebras such that the degree of op_i equals that of op'_i for every i . A *homomorphism* from A to A' is a set of functions $f: K \rightarrow K'$ and $f_i: op_i \rightarrow op'_i$ such that for every x, y, z in K ,

- $f(op_i(x)) = op'_i(f(x))$ if op_i is unary
- $f(x op_i y) = f(x) op'_i f(y)$ if op_i is binary
- $f(op_i(x, y, z)) = op'_i(f(x), f(y), f(z))$ if op_i is ternary; and so on

Therefore, where X is of data type Y refers the concept of homomorphism between X and Y such that X has the same property of Y . For example, where the variable (algebra) AGE is of data type (algebra) $INTEGER$ means that operations, such as addition and subtraction, can be performed on AGE just like those on $INTEGER$. Other user-defined data types can be constructed in a similar manner. For some particular applications, appropriate homomorphism between geometric algebras and geographically algebraic objects will need to be made.

Additionally, like the concept of composition in set theory, a complex algebra could be constructed from several simple algebras. That is, let $\langle \alpha, \bar{\Sigma}_\alpha \rangle$ and $\langle \beta, \tilde{\Sigma}_\beta \rangle$ be two algebras, the algebra $\langle \mathcal{S}, \Sigma_s \rangle$ is a *composition* of $\langle \alpha, \bar{\Sigma}_\alpha \rangle$ and $\langle \beta, \tilde{\Sigma}_\beta \rangle$, such that $\mathcal{S} = \alpha \cup \beta$ and $\Sigma_s = \bar{\Sigma}_\alpha \cup \tilde{\Sigma}_\beta$. The algebra $\langle \mathcal{S}, \Sigma_s \rangle$ is a *super-algebra* of $\langle \alpha, \bar{\Sigma}_\alpha \rangle$ and $\langle \beta, \tilde{\Sigma}_\beta \rangle$, where $\mathcal{S} \supseteq \alpha$, $\mathcal{S} \supseteq \beta$, $\Sigma_s \supseteq \bar{\Sigma}_\alpha$, and $\Sigma_s \supseteq \tilde{\Sigma}_\beta$.

5.2. Primitive geometric algebras

Points, lines and polygons are geometric primitives in GIS. However, only the algebraic constructions of points and lines are introduced in this sub-section. As to the construction of polygons, it involves complex signature constructions and will therefore be introduced later. Please note that here points and lines are mentioned at a conceptual level without referring to their vector or raster implementations.

A simple way to construct the algebra $POINT$ of points is to directly define the signature $\mathbf{p} = \{P_1, P_2, \dots, P_n\}$, where P_i ($i = 1, \dots, n$) are functions of arity 0. The term algebra $\mathbf{T}_{\mathbf{p}}$ associated with the signature $\mathbf{p}$ is exactly $\{P_1, P_2, \dots, P_n\}$. In other words, it implies that the carrier can be constructed from functions with arity 0 by its own signature. As a matter of fact, any set can be constructed in this way.

We may extend the algebra $POINT$ to include function *Distance*. The more complex algebra $POINT$ of points is a tuple $\langle \mathbf{K}, \mathbf{p}, A_p \rangle$, where (a) the carrier $\mathbf{K} = \{REAL, POINT\}$ has two sorts, the real number *REAL* and the set of points *POINT*, and (b) the signature $\mathbf{p} = \{Distance\}$ defines a function returning the distance between two points.

Distance: POINT × *POINT* → *REAL*

The term algebra associated with the algebra $POINT$ of points has the form of $\{r, p, Distance(p_i, p_j)\}$, where $r \in REAL$, $p, p_i, p_j \in POINT$. Furthermore, $\forall i, j \exists d \in REAL, Distance(p_i, p_j) = d$. Please note that the preceding phrase is in fact a mandatory property which the function *Distance* should obey. Although here we specify the property in an English sentence, it is no problem to rewrite it in a mathematical way. However, we have at least two reasons to specify it in English: simplicity and leaving the definition of distance undecided at this moment. As a

matter of fact, the concepts of distance could be defined in many ways, such as Euclidean, Manhattan, Minkowski, Hausdorff and transportation distances (Preparata and Shamos 1985).

Based on the algebra *POINT* of points and letting *LINE* be a set of lines, the algebra *LINE* of lines can be defined by the tuple $\langle \text{LINE}, A_L \rangle$, where carrier $\text{LINE} = \text{POINT} \times \text{POINT}$, signature $A_L = \{\text{Make}, \text{Length}, \text{Intersect}\}$, and $A_L \supset A_P$, where

Make: $\text{POINT} \times \text{POINT} \rightarrow \text{LINE}$ makes a line segment by two endpoints.

Length: $\text{LINE} \rightarrow \text{REAL}$ measures the length of a line.

Intersect: $\text{LINE} \times \text{LINE} \rightarrow \text{POINT}$ returns the intersection point of two lines.

The term algebra of a line is of the form $\text{Make}(p1, p2)$, where $p1, p2$ are two end points. Please also note that, since line algebra inherits the carrier and the signature from the point algebra, it is well defined to express a term like $\text{Distance}(p, \text{Intersect}(l1, l2))$, which is a sequence of functions and corresponds to the distance between point p and the intersection point of lines $l1$ and $l2$.

As mentioned earlier, an appropriate homomorphism should be made between geometric and geographic objects. An example of modelling road systems demonstrates the mechanism of homomorphism. At first, the algebra $\langle \{\text{ROAD}, \text{LOCATION}\}, \{\text{Make_Road}, \text{Intersect}\}, _ \rangle$ is specified to represent the geometric aspect of *ROAD*. Then, the homomorphism between carriers and signatures of these two objects is defined as follows.

$f1: \text{LOCATION} \rightarrow \text{POINT}$

$f2: \text{ROAD} \rightarrow \text{LINE}$

$f3: \text{Make_Road} \rightarrow \text{Make_Line}$

$f4: \text{Intersect}_{\text{ROAD}} \rightarrow \text{Intersect}_{\text{LINE}}$

$f5: \text{Length}_{\text{ROAD}} \rightarrow \text{Length}_{\text{LINE}}$

As a result, any operations upon *ROAD* can be performed in terms of the algebra *LINE*. For example, objects $r1$ and $r2$ of roads can be constructed as $\text{ROAD}(p1, p2)$ and $\text{ROAD}(p3, p4)$ by mimicking the construction of geometric objects of *LINE*. Furthermore, the length and the intersection location can be calculated as follows.

$\text{Length}(r1) = \text{Length}(\text{ROAD}(p1, p2)) = \text{Length}(\text{LINE}(p1, p2))$

$\text{Intersect}(r1, r2) = \text{Intersect}(\text{LINE}(p1, p2), \text{LINE}(p3, p4))$

5.3. First Order Logic

It is well known that propositional logic is a Boolean algebra. As a matter of fact, the first order logic (FOL) is also an algebra. Therefore, expressions of logic can be used in algebraic specification in a coherent manner.

An algebra for the FOL is constructed by assigning Boolean constants to functions corresponding to logic well-formed formulas in the following manners.

- For predicate $P(a_1, a_2, \dots, a_n)$, $a_i \in D_i$, there is a corresponding function,
 $P: D_1 \times D_2 \times \dots \times D_n \rightarrow \text{BOOL}$.
- Define FOL operators as functions in the signature with appropriate axioms.
- For $\forall x ()$, $\forall$ is a mandatory property.
- For $\exists x ()$, the statement, $\{x \mid \text{ } = \text{TRUE}\} \neq \emptyset$, is a mandatory property.

5.4. Object-oriented mechanism

The reason why object-oriented data models attract so much attention is that it has many good properties, such as encapsulation, inheritance, unique behavior of each object, and five abstraction mechanisms, including classification, generalization, specification, aggregation and association (Ghezzi and Jazayeri 1982, Elmasri and Navathe 1989, Date 1990, Andleigh and Gretzinger 1992, Tang *et al.* 1996). It is not difficult to see that algebraic systems also own these capabilities.

In general, an object, a class of objects, and a super-class of object classes can be expressed in terms of algebra. By definition, an algebra is a tuple which encapsulates data, functions and constraints of objects together. The signature specifies the unique set of functions of the algebra and ensures that the right functions are performed upon correct object classes. The mandatory properties enforce the integrity between objects. Furthermore, the mechanism of algebra composition constructs super-class(es) from subclass(es) by encapsulating the data, functions and mandatory properties of subclass(es) and imposing further constraints among subclass(es) as mandatory properties. On the other hand, inheritance is a function from super-class(es) to subclass(es) with preserved properties.

The job of classification can be done by the term algebra with functions of arity 0, and their ranges are sets of elemental objects. Generalization and specification are two kinds of functions with opposite constructions to each other. Generalization maps a power set to a superset, while specification does it the other way. Aggregation is a common function with different types of domains. Association is a constrained function in which elements in domains of different types should have some kinds of the same properties.

5.5. Mixture of operational and declarative statements

The language of data models requires the capability of expressing operational and declarative statements at the same time. In general, the order of the sequence of operational statements is important, while that of declarative statements is immaterial (Chang and Lee 1973, Giarratano and Riley 1989). Fortunately, they can be mixed

appropriately. For instance, a system of simultaneous equations, $f(g(x))=0$ and $h(x)=0$, is considered. The order of these two equations will not affect the solution of x ; however, that in functions $f(g(x))$ and $g(f(x))$ does. Please note that this system of simultaneous equations truly specifies what properties the solution x should meet, but does not mention about how to solve the equations. On the other hand, it also explicitly specifies the operation order of functions f and g . From this simple case, how the conventional algebra handles operational and declarative statements at the same time is demonstrated. As to many sorted algebra, the sequence of functions in the term algebra corresponds to the order of function operations. Additionally, the statements in mandatory properties, which inevitably involve algebraic terms associated with an ordered sequence of functions, could be in any order without affecting the semantics of algebraic systems.

6. Geometric Object Models

6.1. *The family of geometric algebras*

It is commonly taken for granted that spatial objects are composed of points, lines and areas, which may be represented in vector or raster manners. However, this conventional concept is insufficient to handle more complex situations, which include the necessity of exchangeable multi-representation, the capability of possessing composite data types, and the integrity among various objects in the real world, e.g. UIS. To have these capabilities, the technique of algebra is employed to build up a family of geometric models so that more specific and application-oriented geographic objects can be well constructed upon this foundation. Some geometric algebras are shown in the appendix. In the family, there are some key members that portray the organizations and major mechanisms. These key members and their relationships are depicted in Figure 3, where bold arrow lines indicate the construction relationships among algebras and gray lines indicate other functional relationships.

Basically, the family can be tentatively divided into five groups, which are the point group, the line group, the polygon group, the network group and the raster group. However, these five groups are related by some functions. Among them, the point, line and polygon groups are the most essential. Although the structure of these three algebras is discussed here in a mathematical manner, it is however quite natural to implement them in a vector based approach. The network and raster groups are special cases in terms of these three essential algebras. The network group has a special structure of nodes (points) and curved poly-lines (links). The raster group can be treated as a special kind of the point group with discrete coordinates and special functions. That A is constructed from B implies that the carrier and signature of B are also basic constitutions of A . On the other hand, if A and B are solely functionally

related, they are constructed independently but are related to or can be transformed into each other by some functions.

Figure 3. Family of geometric algebras

The whole family originates from the concept of set theory. Firstly, algebras of points and lines are constructed, which has been shown in Section 5.2. Then, algebra *PLINE* of poly-lines can also be constructed straightforwardly. To have the capability of describing curved objects, the algebra *CURVE* is defined for curve objects. Then, the algebras *PLINE* and *CURVE* are combined to construct the algebra *CURVED_PLINE*, which is able to describe roads with curves. For simplicity, the sub-family of the algebras *LINE*, *CURVE*, *PLINE*, and *CURVED_PLINE* is called the line group.

After the algebra *PLINE* and *CURVED_PLINE* are defined, the algebra *POLYGON* may be expected. However, most GIS documents or textbooks describe the concepts of (set of) polygons in a very vague manner and it is necessary to elaborate the concepts in detail. Here, the algebra *POLYGON* is constructed for polygons that are of closed polylines, the algebra *TESSERA* for a set of compact connected polygons, and the algebra *TESSELLATED_ISLANDS* for a group of tesserases that are separated. Therefore, the geometric algebra *POLYGON* can describe the shape of a single area; *TESSERA* for a subdivided area, such as the shapes of city blocks with subdivided lots; and *TESSELLATED_ISLANDS* for geometric shapes of

zoning maps and cadastral maps, where blocks are separated by streets. Furthermore, algebras *TASSELED_REGION*, *TASSELED_TESSERA*, and *TASSELY_TESSELLATED_ISLANDS* are introduced to describe complex shapes, such as geometric aspects of rivers, lakes and transportation lines with station areas, that are composed of lines, polygons, areas and tesseras. The algebras *POLYGON*, *TASSELED_REGION*, *(TASSELED)_TESSERA*, and *(TASSELY)_TESSELLATED_ISLANDS* is called the polygon group.

6.2. Canonical forms

In geometry, canonically mathematical forms of points, lines and polygons are *point(x,y)*, *line(p1,p2)*, and *polygon(p1,p2,p3,...,pn)*, respectively, where *x* and *y* are coordinates, and $p_i(i=1..n)$ are points. Similarly, we may define canonical forms in Table 1 for all the members of the geometric algebras.

Table 1. Canonical forms of geometric algebras

Algebras	canonical forms	notes
point	<i>point(x,y)</i>	x and y are coordinates
line	<i>line(p1,p2)</i>	p1 and p2 are points
curves	<i>curve(p1,p2,...,pn,EQN)</i>	the equation <i>EQN</i> fits all the points $p_1,...,p_n$
plines	<i>pline(p1,p2,...,pn)</i>	
curved_pline	<i>curved_pline(cp1,cp2,...cpn)</i>	cp_i ($i=1..n$) is either a line or a curve
polygon	(1) <i>polygon(p1,p2,...,pn)</i>	(1) line edges
	(2) <i>polygon(cp1,cp2,...cpn)</i>	(2) curved polyline edges
tessera, and the others	(1) <i>tessera({cp1,cp2,...cpn})</i>	(1) edge based
	(2) <i>tessera({(v_i,v_j,EQU_{ij}),...})</i>	(2) vertex based
		(3) The others have the same form structure to that of tessera

The signature includes a set of functions, which can be classified into two categories: entity and operation related. The main purpose of entity related functions is to construct or decompose entities, while that of operation related functions is to define valid operations upon appropriate entities. They will be discussed in the following two sub-sections, respectively.

6.3. Entity related functions

Functions in a form of *Make_X* are constructors, which make up basic entities for the algebra *X*. The term algebra associated with *X* has a very similar structure to that of the corresponding canonical form. For example, the function *Make_Pline* takes a sequence of lines to construct a poly-line. Its term algebra is in the form of *Make_Pline(...(Make_Pline(Make_Line(*p*₁,*p*₂)),*Make_Line*(*p*₂,*p*₃)),...,*Make_Line*(*p*_{*n*-1}, *p*_{*n*}))*, while the canonical form is *pline*(*p*₁,*p*₂,*p*₃,..., *p*_{*n*-1}, *p*_{*n*}). In a sense, the canonical form can be a shorthand of the term algebra, while the term algebra is a possible implementation of the canonical form.

Moreover, constructors could include an indefinite number of attribute domains, such as identifier, name and time. For example, the signature of the algebra *TAGGED_LINE*, which is a kind of line with attributes of identifier, length, time and owner, can be declared as follows.

$$\begin{aligned} \text{Make_Tagged_Line: } & \text{POINT} \times \text{POINT} \times \text{IDENTIFIER} \times \text{LENGTH} \times \text{TIME} \times \text{OWNER} \\ & \rightarrow \text{TAGGED_LINE} \end{aligned}$$

Since the attributes that might be included in the signature are determined case by case, the generic variable *ATTRS* is used to denote the sequence of possible attribute domains. Furthermore, *Q.attr* denotes the value of the particular attribute *attr* of object *Q*. Therefore, *e.identifier* is able to denote the identifier of the line *e*, where *e* ∈ *TAGGED_LINE*.

The attribute *IDENTIFIER* is very special and powerful one. Conventionally, it distinguishes every particular object from others. However, it is also possible to denote the same object which is represented in various ways. For example, a street could be represented by a line at a small scale and a pair of parallel lines at a larger scale. The possible signatures are:

$$\begin{aligned} \text{Make_Single_Line_Street: } & \text{LINE} \times \text{IDENTIFIER} \times \text{SCALE} \rightarrow \text{SINGLE_LINE_STREET} \\ \text{Make_Double_Line_Street: } & \text{LINE} \times \text{LINE} \times \text{IDENTIFIER} \times \text{SCALE} \rightarrow \text{DOUBLE_LINE_STREET} \end{aligned}$$

Since these two objects *SINGLE_LINE_STREET*(*e*₁,*i*,*s*₁) and *DOUBLE_LINE_STREET*(*e*₂,*e*₃,*n*,*s*₂) have the same identifier (name) *i*, they are two different representations of the same street in the real world. Furthermore, it is very typical to construct a polyline by appending line by line in a consecutive manner. However, it is also possible to draw line segments in a random order, but finally conform to the mandatory properties that require all the lines to be connected in a sequence. Therefore, the object construction mechanisms in this work are a few of the possible but typical ways.

Functions in the forms of *Extract_X* are to decompose the object *X* into its components. For example, points and lines can be extracted from polylines, and vertices, edges and polygons from tesserars. In a sense, the mechanisms of constructions and extraction are kinds of reversed operations to each other.

Furthermore, the extraction functions make multi-representation of objects possible. It means that the street edges, for example, can be recognized by extracting edges from zonal city blocks and checking if there is another parallel block edge within a reasonable distance.

6.4. Operation-related functions

Many functions are defined in and between algebras for operational purpose. Some of them are identified below and illustrated in Table 2.

Break(A,B): *A* is a point or of the line group, *B* is of the line group. *B* is broken into two parts by *A*, which, however, remains the same.

In(A,B): *A* is a point, a raster area or of the line group, *B* is a raster area or of the polygon group. It justifies if *A* is in *B*. The returned value is Boolean.

On(A,B): *A* is a point, *B* is of the line group. It justifies if *A* is on *B*. The returned value is Boolean.

Area(A): *A* is of the polygon, the tassel group, or raster areas. It calculates the enclosed or occupied area by *A*.

Maximal_circuit(A): *A* is of the polygon group, the tassel group, or raster area. It returns the outline, shape, edge or boundary of *A* in the form of (a set) of circuits (polygons.)

Line_Intersect(A,B): *A* and *B* are of the line group. It returns the intersection point(s) of *A* and *B*.

Overlay(A,B): *A* and *B* are of the polygon group or the raster area. It returns all the subdivisions in terms of a set of tesserases.

Union(A,B): *A* and *B* are of the polygon group or the raster area. It returns the outlines of the union areas of *A* and *B* in terms of a set of polygons.

Intersect(A,B): It is a general intersection function. *A* and *B* could be points, of the line groups, the polygon group or the raster area. It returns the intersected portions of *A* and *B*.

Buffer(A,B): *A* could be points, of the line groups, the polygon group or the raster area, while *B* is a distance. It returns a set of polygons which are buffers of *A* in a distance of *B*.

Voronoi(A): *A* is a set of points or separated polygons. It returns a tessera where each subdivision is exactly served by a point or a polygon of *A*.

Path(A,B,C): *B* and *C* are two points on the network *A*. It returns a sequence of members of the line group which are links of *A* and connect *B* and *C*.

Flow(A,B,C): *B* and *C* are two sets of starting and ending points on the network *A*. It returns a group of paths with flow from *B* to *C*.

V/R(A): It denotes a group of vector and raster based data conversion functions.

Connect(A): A is a set of cells of raster data. It justifies if components of A are connected altogether.

Slimming(A), Thickening(A,B), Dilation(A,B), Erosion(A,B), Open(A,B), and Close(A,B), Area_Aggregation(A,B): A and B are raster based maps. They also return raster based maps.

Table 2. Inter-algebra functions

algebra function	line group					polygon group							network	raster	number	Boolean
	point	line	pline	curve	curved pline	circuit	polygon	tessera	TI	TP	TT	TTI				
break	•	•	•	•	•											
in	•	•	•	•	•	•	•	•	•	•	•	•		•		•
on	•	•	•	•	•											•
area						•	•	•	•	•	•	•		•	•	
maximal_ circuit						•	•	•	•		•	•		•		
line_ intersect	•	•	•	•	•											
overlay							•	•	•	•	•	•		•		
union							•	•	•	•	•	•		•		
intersect	•	•	•	•	•		•	•	•	•	•	•		•		
buffer	•	•	•	•	•	•	•	•	•	•	•	•		•	•	
Voronoi	•							•	•					•		
path	•	•	•	•	•								•			
flow	•	•	•	•	•								•			
V/R	•	•	•	•	•	•	•	•	•	•	•	•		•		
connect														•		•
slimming														•		
thickening														•		
dilation														•		
erosion														•		
open														•		
close														•		
area aggregation														•		

There are dependencies among functions (see Figure 4), which can be classified into three categories: fundamental, intermediate and advanced ones. The fundamental functions, including *On*, *Line_Intersect*, *Break*, *Extract_edge*, *Extract_Polygon*, *Maximal_Circuit*, and *Distance*, are cornerstones of the other functions. Particularly, the definitions of distances, as mentioned earlier, which might be Euclidean, Manhattan, Minkowski, Hausdorff, or the length of the path on the network, will determine the results and appearances of spatial analyses. On the other hand, the advanced functions, such as *Union*, *Intersect*, *Buffer*, *Voronoi* and *Flow* analysis, are high level functions and very useful to the applications. However, it will be very complicated, not easy to maintain and modify, if one tries to build up these high level functions directly from those fundamental functions. It implies that intermediate functions are needed. The intermediate functions, including *Overlay*, *Shape*, *In* and *Path* are formulated on the basis of the fundamental functions, and common to the advanced functions. Hence, the advanced functions can be finally constructed from common functions that might be fundamental or intermediate.

Raster based functions are quite independent. However, we can still find the three categories of functions here. The fundamental functions include *Connection*, *Slimming*, *Thickening*, *Dilation* and *Erosion*. Based on some of them, Su *et al.* (1997) define the intermediate functions, *Open*, *Close* and others, to construct high level area aggregation operators.

7. Geographical Models and Their Properties

A geographical river could be geometrically described by a sequence of lines. While GIS developers are interested in geographical representations of the real world, people perceive it geographically. However, geometric and geographical descriptions can be linked through the mechanism of homomorphism. A simple example of *ROAD* was given in Section 5.2. Further examples with more complex properties are illustrated in this section.

7.1. Object composition and multi-representation

An object can be perceived to be a composition of several components. For example, water systems are composed of rivers (lines), lakes (polygons) and ports (points). By respectively specifying the homomorphism that associates rivers, lakes and ports with *LINE*, *POLYGON* and *POINT*, the construction of water systems can be made as follows.

Make_Water_System: RIVERxLAKExPORTxATTRS → WATER_SYSTEM

Figure 4. Dependencies among geometric functions

An object can also be represented in multiple ways. For example, another way to specify water systems is to perceive that ports are human facilities, and rivers and lakes are water bodies of algebra *TASSELED POLYGON*. Therefore,

Make_Water_Body: RIVER×LAKE×ATTRS → WATER_BODY

Make_Water_System: WATER_BODY×HUMAN_FACILITY×ATTRS → WATER_SYSTEM

7.2. Networks

In an urban environment, there are many network systems, such as roads, water pipes, power lines and survey monument systems. They share many common characteristics, e.g. connectivity, hierarchical subsystems, flow and routing analysis.

A simplified example of road systems demonstrates how to specify a typical network in an algebraic way.

Suppose that there are three kinds of road subsystems, highways, main streets and local roads. Highways can only be connected to main streets with interchanges. There is no traffic light on the highway, and the distance between consecutive interchanges is at least 10 kilometers. Every intersection on the main streets should have traffic lights. However, on the local roads, the intersections may or may not have traffic lights or stop signs. Every building should be connected to local roads or main streets, but not highways.

In this particular example, a homomorphism is defined that interchanges and intersections are of the algebra *NODE*; traffic lights, stop signs, and buildings are of *POINT*; three road subsystems are of *NETWORK*. The algebra *ROAD_SYSTEM* could be specified as:

$$\text{Make_Road_System: HIGHWAY} \times \text{MAIN} \times \text{LOCAL} \times \text{INTERCHANGE} \times \\ \text{INTERSECTION} \times \text{TRAFFIC_LIGHT} \times \text{STOP} \times \text{BUILDING} \rightarrow \text{ROAD_SYSTEM}$$

Please note that, in this particular algebra *ROAD_SYSTEM*, interchanges, intersections, traffic lights and stop signs are not considered as nodes of the sub-algebras *HIGHWAY*, *MAIN* and *LOCAL*. Instead, they are five kinds of special nodes of the algebra *ROAD_SYSTEM*. There are many mandatory properties that specify the required relationships among the five kinds of special nodes and the three sub-algebras of road systems. Some of them are shown as follows.

$$\begin{aligned} \forall i \in \text{INTERCHANGE}, \exists h \in \text{LINK} \in \text{HIGHWAY}, \exists m \in \text{LINK} \in \text{MAIN}, \\ \text{On}(i, h) \wedge \text{On}(i, m). & \quad (\text{prop.7.2.1}) \\ \forall i \in \text{INTERCHANGE}, \nexists e \in \text{LINK} \in \text{LOCAL}, \text{On}(i, e). & \quad (\text{prop.7.2.2}) \\ \forall i, j \in \text{INTERCHANGE}, \text{Distance}(i, j) \geq 10 \text{ kilometers}. & \quad (\text{prop.7.2.3}) \\ \forall h \in \text{HIGHWAY}, \nexists t \in \text{TRAFFIC_LIGHT}, \text{On}(t, h). & \quad (\text{prop.7.2.4}) \\ \forall h \in \text{HIGHWAY}, \nexists s \in \text{STOP}, \text{On}(s, h). & \quad (\text{prop.7.2.5}) \\ \forall i, j \in \text{LINK} \in \text{MAIN}, \forall k \in \text{LINK} \in \text{LOCAL}, \exists t \in \text{TRAFFIC_LIGHT}, \\ \text{IF } \text{Intersect}(i, j) \text{ THEN } \text{On}(t, \text{Intersect}(i, j)). & \\ \text{IF } \text{Intersect}(i, k) \text{ THEN } \text{On}(t, \text{Intersect}(i, k)). & \quad (\text{prop.7.2.6}) \\ \forall s \in \text{STOP}, \exists i, j \in \text{LINK} \in \text{LOCAL}, \text{On}(s, \text{Intersect}(i, j)). & \quad (\text{prop.7.2.7}) \\ \forall s \in \text{STOP}, \nexists m \in \text{LINK} \in \text{MAIN}, \text{On}(s, m). & \quad (\text{prop.7.2.8}) \\ \forall b \in \text{BUILDING}, (\exists i \in \text{LINK} \in \text{LOCAL}, \text{Access}(b, i)) \vee \\ (\exists j \in \text{LINK} \in \text{MAIN}, \text{Access}(b, j)) & \quad (\text{prop.7.2.9}) \end{aligned}$$

Property 7.2.1 ensures that interchanges connect highways and main streets, while property 7.2.2 disallows local roads to be connected to highways. Properties 7.2.3, 4 and 5 check that the distances between interchanges are greater than 10 kilometers

and there is no traffic light or stop sign on highways. Property 7.2.6 ensures that every intersection of main streets with another main street or local roads has traffic lights. However, since it is not required that every intersection of local road has traffic lights or stops signs, properties 7.2.7 and 8 instead specify that every stop sign should be placed on local roads only. Property 7.2.9 ensures that every building is able to access a main street or a local road.

7.3. Integrity

Owing to various requirements, different departments in a government realize, describe and represent the urban system in different ways. For example, the department of urban development controls land use by zoning maps, which are composed of polygons, and maintains monument maps, which are points and lines. On the other hand, the department of transportation monitors and analyzes traffic flow on street maps, which are networks. These maps are highly related to each other. Any change on a map might require corresponding modification on other maps. In other words, data integrity among maps of different representations should be conformed. The geographic models of these illustrative examples are shown below.

Zoning map is of *TESSELLATED_ISLAND*. Blocks, which are of *TESSERA*, are separated by streets. Land use zones in blocks are represented by polygons within tessellated blocks. Monument maps, which are of *NETWORK*, and include survey control points and central lines of roads, are complementary to zoning maps. These two kinds of maps are highly interdependent. For example, the centre line of a road should be parallel and equal distance from the two block edges that define the road. This mandatory property can be expressed as follows (see prop.7.3.1).

$$\begin{aligned}
 &\forall c \text{ (of } \textit{CENTRAL_LINE}) \in n \text{ (of } \textit{MONUMENT_MAP}), \\
 &\exists e1, e2 \in E \text{ (of set of } \textit{CURVED_PLINE}) \in m \text{ (of } \textit{ZONING_MAP}) \\
 &\quad E = \textit{Extract_Edge}(\textit{Extract_Tessera}(m)), \\
 &\textit{Parallel}(e1, c) \wedge \textit{Parallel}(e2, c) \wedge (\textit{Distance}(e1, c) = \textit{Distance}(e2, c)). \\
 &\text{(prop.7.3.1)}
 \end{aligned}$$

On the other hand, every road defined by two parallel block edges should also have a central line on the monument map. Property 7.3.2 below gives the mandatory property. Please note that the set E of block edges is defined by extracting all the polylines from the zoning map and finding the maximal circuits. Exploding the circuits into curved polylines is to conform to the agreement of data types in the function of parallelism checking.

$$\begin{aligned}
 &\forall e1, e2 \in E \text{ (of set of } \textit{CURVED_PLINE}) \in m \text{ (of } \textit{ZONING_MAP}) \\
 &\quad E = \textit{Explode}(\textit{Maximal_Circuit}(\textit{Become_Curved_Pline}(\textit{Extract_Edge}(\textit{Extract_Tessera}(m))))), \\
 &\exists c \text{ (of } \textit{CENTRAL_LINE}) \in n \text{ (of } \textit{MONUMENT_MAP}),
 \end{aligned}$$

IF (Distance($e1, e2$) <= 50 meters) THEN Parallel ($e1, c$) $\wedge$ Parallel($e2, c$).
(prop.7.3.2)

7.4. Spatial query and analysis

Query and analysis are two kinds of common services that databases offer. Although they are eventually implemented as operational statements, they are also possible to be specified in a declarative manner. In terms of geometric algebra, they are operational related functions. In general, queries involves data retrieval and simple arithmetical calculations, while spatial analysis involve more complex geometric computations. For instance, to query the residential and commercial areas in zoning map m is to evaluate the expression

$$\begin{aligned} & \text{Area}(m, m.\text{use} = \text{"residential"} \vee m.\text{use} = \text{"commercial"}) \\ &= \sum_{\substack{z.\text{use} = \text{"residential"} \\ z.\text{use} = \text{"commercial"}}} \text{Area}(z), \quad \text{where } z \in Z, Z = \text{Extract_Polygon}(b), \forall b(\text{of BLOCK}) \in m. \end{aligned}$$

On the other hand, buffering and Voronoi diagram analysis are two common functions in UIS when service levels of public facilities are evaluated. Both of these two analyses overlay service areas of public facilities with residential and commercial land uses in the zoning map m , and return a set of a pair of items, facility identifier and service level. However, the two methods of computing service areas are different. One is to calculate the buffer areas surrounding public facilities within a certain distance d . However, the buffer areas might not cover the whole area under consideration. The other is based on the algorithm of Voronoi diagram computation, which divides the whole area into subdivisions, to calculate the service areas. Each subdivision includes exactly one facility that is the closest place to provide service for the subdivision. It is also true that every facility is bounded by a subdivision. However, the service distance might be great due to scarce facilities.

$$\text{Buffering_Service: ZONING_MAP} \times \text{FACILITY} \times \text{DISTANCE} \rightarrow 2^{\text{INTEGER} \times \text{REAL}}$$

$$\text{Voronoi_Service: ZONING_MAP} \times \text{FACILITY} \rightarrow 2^{\text{INTEGER} \times \text{REAL}}$$

$$\text{Buffering_Service}(m, f, d) =$$

$$\begin{aligned} & \{ (s.\text{id}, v) \mid \forall s \in \text{Extract_Polygon}(m, m.\text{use} = f) \\ & \quad r = \text{Extract_Polygon}(m, m.\text{use} = \text{"RESIDENTIAL"} \vee m.\text{use} = \text{"RESIDENTIAL"}) \\ & \quad v = \text{Area}(s) / \text{Intersect}(\text{Buffer}(S, d), r) \} \end{aligned}$$

$$\text{Voronoi_Service}(m, f, d) =$$

$$\begin{aligned} & \{ (s.\text{id}, v) \mid T = \text{Voronoi}(\text{Extract_Polygon}(m, m.\text{use} = f)), \\ & \quad \forall s \in \text{Extract_Polygon}(m, m.\text{use} = f), \\ & \quad \exists t \in \text{Extract_Polygon}(T), \text{Within}(s, t), \\ & \quad r = \text{Extract_Polygon}(m, m.\text{use} = \text{"RESIDENTIAL"} \vee m.\text{use} = \text{"RESIDENTIAL"}) \\ & \quad v = \text{Area}(s) / \text{Intersect}(t, r) \} \end{aligned}$$

7.5. Integration of vector and raster based operations

The differences between vector and raster based representations might not be as big as they look. Some of these differences are that the coordinates of vector based representations are continuous and their data structures could be in any form, but those of raster based are discrete and in a grid data structure. However, in the digital computer world, any representation is discrete, and the grid system is one of the possible data structures. Therefore, they do not have any difference in a pure theoretical sense. However, they are indeed different when the conceptual systems are implemented in computer codes, data are gathered from different devices, and programs are applied to the real world. The vector based systems gain a very high degree of precision of object representations and computations, by highly approximating the mechanism of conventional geometry that represents the real world in a continuous way. On the other hand, raster based systems take lower resolutions to represent the outside world. However, in many circumstances the computational time and storage of raster systems are more efficient than those of the vector based approach, while the degrees of precision and accuracy are still acceptable.

Many land use structure analyses in UIS are performed in raster form to save computational time with reasonable accuracy. However, topological and zoning maps, which are important data sources for spatial analyses, are digitized in vector form. Therefore, functions of vector to raster conversion are needed. Please note that this not only performs geometric conversions but also attribute allocations. For example, a large land use zone could be divided into several raster cells. The population associated with the large zone has to be allocated to each cell according to appropriate criteria. These allocation rules could be expressed as a part of mandatory properties to ensure that the intentions of planners are correctly implemented. In the algebra *Allocation* below, the population allocated to the raster cell (i,j) , which is within vector zone m , is according to its area proportion. More complicate population allocation methods and situations of boundary cells can be handled and specified in a similar way.

$$\begin{aligned}
 & \text{Allocation: } V_ZONE \times \text{INTEGER} \times \text{INTEGER} \rightarrow R_ZONING \\
 & \text{Allocation}(m,i,j).population \\
 & = R_ZONING_MAP(i,j).population \\
 & = m.population \times [Area(CELL(i,j)) / Area(V_ZONING_MAP(m), within(CELL(i,j),m))]
 \end{aligned}$$

8. Conclusions

This work demonstrates that many sorted algebra is an appropriate language for spatial data models. Being carefully constructed, geometric algebra can describe

well-defined entities, functions, constraints, and relations of vector and raster based objects. Then, some geographic data models can be illustrated, by making homomorphisms to specify particular properties and the tight interdependency among multiple representations of urban information systems in different departments.

Developing GIS data models in algebraic form is still in its infancy. Further efforts, including issues of language completeness, efficiency, data structure, consistency checking and proving, must be done in the future. As expected, many sorted algebra provides a common language for end users to precisely specify their requirements and for system developers to correctly implement systems in an efficient manner.

Although this work elaborates on the practical nature of algebraic data models in a rigid mathematical form, data model languages can also be expressed in other forms, e.g. diagrams, graphs, plain sentences, program-like statements and pseudo-codes. Each form expresses conceptual data models with different degrees of communicability, understanding, abstraction and preciseness. Therefore, any combination of expressive forms should be freely employed if deemed necessary and appropriate in practice.

Appendix

POLYGON

Signature

Make_Polygon: CURVED_PLINE $\times$ ATTRS $\rightarrow$ POLYGON

Extract_Edge: POLYGON $\rightarrow$ CURVED_PLINE

Extract_Vertices: POLYGON $\rightarrow$ 2^{POINT}

Within: POLYGON $\times$ POLYGON $\rightarrow$ BOOL

Mandatory Properties

POLYGON($p_1, p_2, \dots, p_n$) denotes Make_Polygon(CURVED_PLINE($p_1, p_2, \dots, p_n$), attrs).

Extract_edge(Make_Polygon(CURVED_PLINE($p_1, p_2, \dots, p_n$), attrs)) =

Extract_Pline(CURVED_PLINE($p_1, p_2, \dots, p_n$))

Extract_Curve(CURVED_PLINE($p_1, p_2, \dots, p_n$))

Make_Polygon(Extract_Edge(g)) = g

Extract_Vertices(Make_Polygon(CURVED_PLINE($p_1, p_2, \dots, p_n$), attrs)) =

Extract_Point(CURVED_PLINE($p_1, p_2, \dots, p_n$), attrs)

POLYGON($p_1, p_2, \dots, p_n$) = CIRCUIT($p_1, p_2, \dots, p_n$)

Explode(Make_Circuit(Extract_Edge(g))) = Extract(g)

ensures that the polygon g is a circuit.

Within(POLYGON(P), POLYGON(Q)) = Within(CIRCUIT(P), CIRCUIT(Q))

TESSERA

Signature

Make_Tessera: 2^{POLYGON} $\times$ ATTRS $\rightarrow$ TESSERA

Make_Tessera: (2^{LINE} 2^{CURVE}) $\times$ ATTRS $\rightarrow$ TESSERA^{POLYGON}

Extract_Polygon: TESSERA $\times$ COND $\rightarrow$ 2^{POLYGON}

Extract_Edge: TESSERA $\times$ COND $\rightarrow$ 2^{LINE} 2^{CURVE}

Extract_Vertices: TESSERA $\times$ COND $\rightarrow$ 2^{POINT}

Shape: TESSERA $\times$ COND $\rightarrow$ 2^{POLYGON}

Area: TESSERA $\times$ COND $\rightarrow$ REAL returns the total areas satisfying the condition.

Within: TESSERA $\times$ TESSERA $\rightarrow$ BOOL

Mandatory Properties

$$\left\{ \begin{array}{ll} n = |\text{Extract_Vertices}(t)| & \text{returns the number of vertices where a curve is counted as two vertices.} \\ e = |\text{Extract_Edge}(t)| & \text{returns the number of edges where a curve is counted as one} \\ & \text{edge.} \\ r = |\text{Extract_Polygon}(t)| & \text{returns the number of polygons.} \\ n - e + r = 2 & \end{array} \right.$$

$t = \text{TESSERA}(\text{Extract_Polygon}(t))$
 $= \text{TESSERA}(\text{POLYGON make_Polygon}(\text{Become_Curved_Pline}(\text{Extract_Edge}(\text{Extract_Polygon}(t))))))$
 $\text{Shape}(t) = \text{Maxiaml_Circuit}(\text{Become_Curved_Pline}(\text{Extract_Edge}(t)))$
 $\text{Within}(p,q) = \text{Within}(\text{Shape}(p), \text{Shape}(q))$

註解: (1) We cannot omit Extract_Polygon here, because directly applying Extract_Edge will produce a bunch of curved_plines that will be not satisfied with mandatory property of Become_Curve_Pline, which requires only one Curved_Pline.
 (2) We union over all the polygons that are produced by the deepest Extract_Polygon.

OVERLAY

Signature

$\text{Visual_Overlay} : \text{TESSERA} \times \text{TESSERA} \rightarrow 2^{\text{TESSERA}}$
 $\text{One_Way_Overlay} : \text{TESSERA} \times \text{TESSERA} \rightarrow 2^{\text{TESSERA}}$
 $\text{Overlay} : \text{TESSERA} \times \text{TESSERA} \rightarrow \text{TESSERA}$

Mandatory Properties

$\text{Extract_Edge}(P) = A$, where P is a polygon.

$\text{Extract_Edge}(Q) = B$, where Q is a polygon.

$$R = \bigcup_{\substack{a \in A \\ b \in B}} \text{Break}(a,b)$$

$$S = \bigcup_{\substack{a \in A \\ b \in B}} \text{Break}(b,a)$$

$$T = R \cup S$$

$$W = \{ w \mid \text{In}(w, \text{Shape}(P)), w \in S \}$$

$$\text{Overlay}(P,Q) = \text{Make_Tessera}(T)$$

$$\text{Visual_Overlay}(P,Q) = \{P,Q\}$$

$$\text{One_Way_Overlay}(P,Q) = \{ \text{Make_Tessera}(W), Q \}$$

$$\forall t \in \text{Extract_Polygon}(\text{Overlay}(P,Q)),$$

$$t.\text{lineage} = \{P,Q\} \text{ if } \text{Within}(t, \text{Shape}(P)) \wedge \text{Within}(t, \text{Shape}(Q))$$

$$t.\text{lineage} = \{P\} \text{ if } \text{Within}(t, \text{Shape}(P)) \wedge \neg \text{Within}(t, \text{Shape}(Q))$$

$$t.\text{lineage} = \{Q\} \text{ if } \neg \text{Within}(t, \text{Shape}(P)) \wedge \text{Within}(t, \text{Shape}(Q))$$

$$t.\text{lineage} = \{ \} \text{ if } \neg \text{Within}(t, \text{Shape}(P)) \wedge \neg \text{Within}(t, \text{Shape}(Q))$$

BUFFER

Signature

$$\text{Buffer} : 2^{\text{POINT}} \times \text{REAL} \rightarrow 2^{\text{CIRCUIT}}$$

$$\text{Buffer} : 2^{\text{CURVED_PLINE}} \times \text{REAL} \rightarrow 2^{\text{CIRCUIT}}$$

$$\text{Buffer} : 2^{\text{TESSERA}} \times \text{REAL} \rightarrow 2^{\text{CIRCUIT}}$$

Mandatory Properties

$\text{Buffer}(P,d) = B$, where $P = \{p \mid p \text{ is a point}\}$

$$\forall b \in B \in B, \exists p \in P, \text{Distance}(b,p) = d \mid \exists p \in P, \text{Distance}(b,p) < d$$

$\text{Buffer}(L,d) = B$, where $L = \{l \mid l \text{ is a curved poly_line}\}$

註解: using make_polygon(explode(circuit)) to become an area.

註解: reason

$\forall b \in C \in B, \exists l \in L, \text{Distance}(b, l) = d, \exists l \in L, \text{Distance}(b, l) < d$

註解: reason

Buffer(T,d) = B, where T = {t | t is a tessera}

$\forall b \in C \in B, \exists t \in T, \text{Distance}(b, \text{Shape}(t)) = d, \exists t \in T, \text{Distance}(b, \text{Shape}(t)) < d$

註解: shape of tessera

註解: reason

VORONOI

註解: using make_polygon(explode(circuit)) to become an area.

Signature

Voronoi : $2^{\text{POINT}} \rightarrow \text{TESSERA}$

Voronoi : $\text{TESSELLATED_ISLANDS} \rightarrow \text{TESSERA}$

Mandatory Properties

Extract_Polygon(Voronoi(P)) = G, where P = {p | p is a point}

$\forall g \in G, \forall v, \text{In}(v, g), \exists i \in P, \forall j \in P, i \neq j,$
 $\text{Distance}(v, i) \leq \text{Distance}(v, j),$

$g.\text{cover} = 1$

註解: attribute

Extract_Polygon(Voronoi(T)) = G, where T is a tessellated islands.

$P = \{p \mid p = \text{Shape}(t), \forall t \in \text{Extract_Tessera}(T)\}$

$\forall g \in G, \forall v, \text{In}(v, g), \exists i \in P, \forall j \in P, i \neq j,$
 $\text{Distance}(v, i) \leq \text{Distance}(v, j),$

$g.\text{cover} = 1$

註解: attribute

RASTER_ALGEBRA

註解: using make_polygon(explode(circuit)) to become an area.

Signature

Make_Cell: $\text{INTEGER} \times \text{INTEGER} \times \text{ATTRS} \rightarrow \text{CELL}$

Make_Area: $2^{\text{CELL}} \times \text{ATTRS} \rightarrow \text{AREA}$

Make_Layer: $\text{INTEGER} \times \text{INTEGER} \times 2^{\text{CELL}} \times \text{ATTRS} \rightarrow \text{LAYER}$

Make_Rmap: $2^{\text{LAYER}} \times \text{ATTRS} \rightarrow \text{RMAP}$

Distance: $\text{CELL} \times \text{CELL} \rightarrow \text{INTEGER}$

Distance: $\text{AREA} \times \text{AREA} \rightarrow \text{INTEGER}$

Connected: $\text{CELL} \times \text{CELL} \rightarrow \text{BOOL}$

Overlay: $\text{LAYER} \times \text{LAYER} \rightarrow \text{LAYER}$

Buffer: $\text{AREA} \times \text{INTEGER} \rightarrow \text{AREA}$

Voronoi: $\text{LAYER} \times 2^{\text{AREA}} \rightarrow \text{LAYER}$

V2R: $\text{TESSELLATED_ISLANDS} \times \text{INTEGER} \rightarrow \text{RMAP}$

R2V: $\text{LAYER} \rightarrow \text{TESSELLATED_ISLANDS}$

Vbased_RVOverlay: $\text{TESSELLATED_ISLANDS} \times \text{RMAP} \rightarrow \text{TESSELLATED_ISLANDS}$

Rbased_RVOverlay: $\text{TESSELLATED_ISLANDS} \times \text{RMAP} \rightarrow \text{RMAP}$

Mandatory Properties

$\forall$ LAYER(m,n,{CELL(i,j,...),...},...), $1 \leq i \leq m$, $1 \leq j \leq n$.

Distance(CELL(i,j), CELL(u,v)) = $\sqrt{[(i-u)^2 + (j-v)^2]^{1/2}}$

註解: There are many different definitions.

Distance({C1,...,Ci,...}, {D1,...,Dj,...}) = $\min_{i,j}$ Distance(Ci,Dj)

IF Distance(c1,c2)=1 THEN Connected(c1,c2).

IF (Connected(a,b) $\wedge$ Connected(b,c)) THEN Connected(a,c)

$\forall u,v \in \text{AREA}(\{...,u,...,v,...\})$, Connected(u,v).

註解: Area has to be connected.

Buffer(A,d) = { p | $\exists a \in A$, Distance(p,a) $\leq$ d }

Voronoi(Y,{A1,...,Ai,...}) = $\bigcup_i B_i = Y$, where

註解: Different definitions of distances will give different buffer shapes.

$\forall u,v \quad B_u \cap B_v = \{ \}$,

$\forall b_i \in B_i, \exists a_i \in A_i, \forall a_j \in A_j, i \neq j, \text{Distance}(a_i,b_i) \leq \text{Distance}(a_j,b_j)$,

$b_i.\text{cover} = 'A_i'$

References

- ANDLEIGH, P. K., and GRETZINGER, M. R., 1992, *Distributed object-oriented data-systems design* (Englewood Cliffs: Prentice-Hall).
- BREU, R., 1991, Algebraic specification techniques in object-oriented programming environments. In *Lecture notes in computer science* 562 (New York: Springer-Verlag).
- CAR, A., and FRANK, A. U., 1995, Formalization of conceptual models for GIS using GOFER. *Computers, Environment, and Urban Systems*, **19**, (2), 89-98.
- CATTELL, R. G. G., 1991, *Object data management – object-oriented and extended relational database systems* (Reading: Addison-Wesley).
- CHANG, C. L., and LEE, R. C. T., 1973, *Symbolic logic and mechanical theorem proving* (San Diego: Academic Press).
- CHEN, F. J., 1997, *The development of an integrated geographic database with persistent programming languages*. Master thesis, (Tainan, Taiwan: National Cheng Kung University).
- COUCLELIS, H., 1997, From cellular automata to urban models: new principles for model development and implementation. *Environment and Planning B: Planning and Design*, **24**, 165-174.
- COWEN, D. J., and SHIRLEY, W. L., 1991, Integrated planning information systems. *Geographical Information Systems*, **2**, 297-310.
- DATE, C. J., 1990, *Relational database* (Reading: Addison-Wesley).
- ELMASRI, R., and NAVATHE, S. B., 1989, *Fundamentals of database systems* (Redwood City: Benjamin/Cummings).
- FUTASUGI, K., GOGUEN, J. A., KOUANNAUD, J. P., and MESEGUER, J., 1985, Principles of OBJ2. In *Proceedings 1985 ACM Symposium on Principles of Programming Languages*, NY, pp.52-66.
- GHEZZI, C., and JAZAYERI, M., 1982, *Programming language concepts* (New York: John Wiley).
- GIARRATANO, J., and RILEY, G., 1989, *Expert systems* (Boston: PWS-Kent Publishing).
- GOGUEN, J., THATCHE, J., and WAGNER, E., 1976, An initial algebraic approach to the specification, correctness, and implementation of abstract data types. In *Research Report of IBM TJ Watson Research Center* (Armonk, NY: IBM).
- GOODCHILD, M. F., 1991, The technological setting of GIS. In *Geographical Information Systems*, edited by Maguire, D. J., Goodchild, M. F., & Rhind, D. W., (London: Longman), pp. 45-54.
- GRAHAM, I., 1991, *Object-oriented methods*. Addison Wesley.

- GUTING, R. H., 1988, Geo-relational algebra: a model and query language for geometric database systems. In *Lecture Notes in Computer Science in Advances in Database Technology(EDBT'88)*, edited by J. Schmidt, S. Ceri and M. Missikoff (Berlin: Springer-Verlag), pp.506-527.
- HENNESSY, M., 1988, *Algebraic theory of processes* (London: MIT press).
- JAFFAR, J., LASSEZ, J. L., and MAHER, M. J., 1986, Some issues and trend in the semantics of logic programming. In *Research Report of IBM TJ Watson Research Center* (Armonk: IBM).
- KAINZ, W., EGENHOFER, M. J., and GREASLEY, I., 1993, Modelling spatial relations and operations with partially ordered sets. *Int. J. of GIS*, **7**, (3), 215-229.
- KOSTERS, G., PAGEL, B.-U., SIX, H.-W., 1997, GIS-application development with GEOOOA. *Int. J. Geographical Information Science*, **11**, (4), 307-335.
- LAURINI, R., and THOMPSON, D., 1992, Fundamentals of spatial information systems. In *The APIC series*, 37 (London: Academic Press), pp.95-99.
- LIN, F. T., 1989, *MATHCORE: A mathematical constraint resolution system*. Ph.D. dissertation (Chicago: Northwestern University).
- LIN, F. T., and YOUNG, J. J., 1996, An algebraic data model for GIS. In *Proceedings of GIS in ASIA, Selected papers of the Asia GIS/LIS AM/FM and Spatial Analysis Conference*, pp.361-378.
- MAGUIRE, D. J., and DANGERMOND, J., 1991, The functionality of GIS. In *Geographical Information Systems*, edited by D. J. Maguire, M. F. Goodchild, & D. W. Rhind (London: Longman), pp.319-335.
- PEUQUET, D. J., 1984, A conceptual framework and comparison of spatial models. *Cartographica*, **21**, 66-113.
- PIWOWAR, J. M., LeDREW, E. F., and DUDYCHA, D. J., 1990, Integration of spatial data in vector and raster formats in a geographic information system environment. *Int. J. Geographical Information Systems*, **4**, (4), 429-444.
- PREPARATA, F. P., and SHAMOS, M., 1985, *Computational geometry: an introduction*.
- PRESSMAN, R. S., 1993, *Software engineering -- a practitioner's approach*. (New York: McGraw-Hill).
- RITTER, G. X., WILSON, J., and DAVIDSON, J., 1990, Image algebra: an overview. *Computer Vision, Graphics, and Image Processing*, **49**, 297-331.
- SAHNI, S. (1985) *Concepts in discrete mathematics*. 2nd ed. (the Camelot Publishing Co.)
- STINY, G., 1991, The algebras of design. *Research in Engineering Design*, **2**, 71-181.

- SU, B., LI, Z., LODWICK, G., and MULLER, J.-C., 1997, Algebraic models for the aggregation of area features based upon morphological operators. *Int. J. Geographical Information Science*, **11**, (3), 233-246.
- TAKEYAMA, M., and COUCLELIS, H., 1997, Map dynamics: integrating cellular automata and GIS through geo-algebra. *Int. J. Geographical Information Science*, **11**, (1), 73-91.
- TANG A. Y., ADAMS, T. M., and USERY, E. L., 1996, A spatial data model design for feature-based geographical information systems. *Int. J. GIS*, **10**, (5), 643-659.
- TOMLIN, C. D., 1991, Cartographic modelling. In *Geographical Information Systems: Principles and Applications*, edited by D. J. Maguire, M. F. Goodchild, and D. W. Rhind (London: Longman), pp.361-374.
- ULLMAN, J. D., 1982, *Principles of database systems*. 2nd ed. (Rockvill: Computer Science Press).
- WAGNER, D. F., 1997, Cellular automata and geographic information systems. *Environment and Planning B: Planning and Design*, **24**, 219-234.