Sheaf Data Model: Context and Overview

David M. Butler

Limit Point Systems, Inc.

1 The notion of a data model

A data model is a theory describing computer data. The term was introduced by E.F. Codd in the early 1970's to describe the relationship between previous approaches to data management and the relational data model which he had just introduced. Formally, a data model specifies three things:

- a class of mathematical objects which are used to model data
- the operations on those objects
- the constraints between the objects that must be satisfied in a valid database

The purpose of a data model is to serve as a basis for analysis, design, and implementation of database management systems (DBMS). That is, a DBMS will implement in software (or sometimes in hardware) the operations of the model which will allow clients of the system to store and manipulate their data as instances of the objects of the model.

2 The relational data model

The best known and by far the most successful data model is the relational data model. Currently all major DBMS's (Oracle, Informix, Sysbase, ...) are based on some form of the relational model. To the commercial data management industry, data management is essentially indistinguishable from relational database management system (RDBMS) technology.

In the relational data model, the mathematical objects are relations on domains and the operations are given by the relational algebra. The terms relation, domain, and relational algebra have detailed, rigorous definitions in mathematics and we will need to understand these concepts in order to understand the relational model and its limitations. However, we can avoid the mathematical details by using the table analogy that is widely used in the practical database world.

2.1 Objects in the relational data model

A mathematical set is any collection of objects, entities, or anything else we wish to think about. A domain is a set of values that can be directly represented on the computer, in

other words a computer data type. Three very common domains are integer numbers, real numbers, and character strings. In the table picture, a domain is a table with a single column that lists all possible values in the domain, as in Figure 1. The name of the domain is the heading for the column. We've chosen the number of values in this domain to be very small in order to make the table easy to draw, in practice the number of values would be much larger.

TINY_INT
0
1
2

Figure 1: A very small domain in table form.

The binary Cartesian product, $A \times B$, of two sets A and B is a set which contains all possible pairs (a,b), where a is a member of A and b is a member of B. In the table picture, it is a table with two columns, one for A and one for B, as in Figure 2. This table shows the Cartesian product of the domain TINY_INT with itself. Each row in this table contains a pair of values and hence corresponds to a member of the product set. Each column corresponds to one of the factors in the product.

TINY_INT × TINY_INT	
TINY_INT	TINY_INT
0	0
0	1
0	2
1	0
1	1
1	2
2	0
2	1
2	2

Figure 2: The Cartesian product TINY_INT × TINY_INT in table form.

The notion of Cartesian product can be extended to more than just two factors. The n-ary Cartesian product $A \times B \times C \times \ldots$ (n factor sets) is a table with n columns, one for each factor. Each row contains n values, one from each one of the factors and there is a row in the table for each possible combination of values. Each row is called an n-tuple and the n-ary Cartesian product is the set of all such n-tuples.

A relation is a subset of a Cartesian product set. It can be viewed as a table with the same column headings as the Cartesian product it is a subset of, but only *some* of the rows. It is called a relation because we can select the subset to represent all those pairs that satisfy some relationship between the two columns. For instance, in Figure 3, we show the relation LESS_THAN_OR_EQUAL:

LESS_THAN_OR_EQUAL	
TINY_INT	TINY_INT
0	0
0	1
0	2
1	1
1	2
2	2

Figure 3: The relation LESS_THAN_OR_EQUAL in table form.

A relation schema or relation type is the list of column headings for the table or, equivalently, the list of factors in the Cartesian product which the relation is a subset of. There are many different possible subsets of the rows of a given Cartesian product set and hence there are many possible relations for a given relation type. When we want to emphasize that we a talking about a specific subset of the rows of a given relation type, we will use the term relation instance.

2.2 Representing application data as relations

2.2.1 Entities and relationships

Applications are often analyzed for data base purposes using the dual notions of entity and relationship. An entity is any thing or object in the real world which is distinguishable from all other objects. Entities have attributes. An attribute is a named property that takes its value from some domain. An entity is represented by its set of attribute values; the attribute values identify the entity and describe its state. A relationship is an association between entities.

When the relational model is used to store application data, the application data is typically organized so that a relation represents either an entity in the application or a relationship between entities.

2.2.2 A simple example

For instance, in a personnel application we might have an EMPLOYEE table and a MANAGED_BY table. The EMPLOYEE table , Figure 4, is an entity table. Each row in

the table represents an entity (an employee) and the columns in the table represent attributes of the entity, for instance employee_id, name, job title, and salary.

EMPLOYEE			
ID	NAME	TITLE	SALARY
1	John Jones	Programmer	$100,000
2	Mary Smith	President	$120,000
3	Michael Brown	Programmer	$80,000
4	Linda Green	Secretary	$50,000

Figure 4: The EMPLOYEE entity relation

The MANAGED_BY relation, Figure 5, is a relationship relation. Each row in the table represents the relationship between two employees, one being the manager of the other. The columns in this table contain the ids of the relevant employees.

MANAGED_BY	
WORKER	MANAGER
1	2
3	2
4	2

Figure 5: The MANAGED_BY relationship relation

2.2.3 The HAS_A relationship

Since an entity is any thing or object, one can consider an attribute value to be an entity itself. For instance, a name can be considered an entity. From this point of view, the entity-attribute association can be considered a relationship between two entities, the primary entity and the attribute entity. This very fundamental relationship is often called the HAS_A relationship. The HAS_A relationship is built into the relational data model; it is directly represented by the relationship between a table and its columns. Other relationships, such as the MANAGED_BY relationship in the example above, must be represented by additional tables.

2.3 Operations in the relational model

There are a large number operations that one can perform on relations. These operations take one or more relations as input and produce a relation as output. Equivalently, the operations take one or more tables as input and produce a table as output. These

operations are not all independent of each other, some can be implemented using the others. It is customary to pick an independent subset and refer to this fundamental subset as the operators of the relational algebra, but there is more than one choice of the fundamental subset. For our purposes, the 6 fundamental operators in the relational algebra are:

- Cartesian product
- selection
- projection
- union
- intersection
- rename

We have already described the Cartesian product operator. We will describe each of the others in turn, using the table analogy.

2.3.1 Selection

The selection operator takes a table and a row selection condition and returns a table containing only the rows that match the selection condition. For instance,

SELECT rows with SALARY >= $100,000 in relation EMPLOYEE

returns Figure 6. Note that Figure 6 does not have a name. As we'll see below, the rename operator allows us to give a table a name. But if the table produced by an operator is just a temporary result, to be used only as input to another operator, there is no need for it to have a name.

ID	NAME	TITLE	SALARY
1	John Jones	Programmer	$100,000
2	Mary Smith	President	$120,000

Figure 6: Result of the first selection operator example.

Another example of selection, which we will use shortly is:

SELECT rows with TITLE = Programmer in relation EMPLOYEE

which returns Figure 7.

EMPLOYEE			
ID	NAME	TITLE	SALARY
1	John Jones	Programmer	$100,000
3	Michael Brown	Programmer	$80,000

Figure 7: Result of the second selection operator example

2.3.2 Projection

The projection operator is similar to the selection operator, except it works on columns. The project operator takes a table and a list of column names as input and produces a table which has only the named columns as output. Since two rows may have differed only in one of the columns that was removed, the result may have duplicate rows. If so, only one of the duplicate rows is retained, the others are discarded. As an example,

PROJECT columns named NAME in relation EMPLOYEE

produces the list (i.e. table) of all the employees names, as shown in Figure 8.

NAME
John Jones
Mary Smith
Michael Brown
Linda Green

Figure 8: Result of the projection operator example

2.3.3 Union

The union operator takes two tables as input and creates as output a table that has all the rows that are in either of the input tables. The union operator can only be used on tables which both have the same relation type (column headings). As an example, we can combine the results of the two select operations above with the following operation:

UNION relation Figure 6 with relation Figure 7

which produces Figure 9 as a result.

ID	NAME	TITLE	SALARY
1	John Jones	Programmer	$100,000
2	Mary Smith	President	$120,000
3	Michael Brown	Programmer	$80,000

Figure 9: Result of union operation

2.3.4 Intersection

The intersection operator takes two tables as input and creates as output a table containing all rows that are in both tables. Like the union operator it can only be used on tables which both have the same relation type. So we can also apply the intersection operator to Figure 6 and Figure 7

INTERSECT relation Figure 6 with relation Figure 7

which produces Figure 10 as a result.

ID	NAME	TITLE	SALARY
1	John Jones	Programmer	$100,000

Figure 10: Result of intersection operation

2.3.5 Rename

The operators described above all produce nameless tables. If we want to refer to a resultant table by name, we have to give it a name. The rename operator does this.

2.3.6 Other operators

The set of operators we have described above is a so-called "primitive" set. It is a minimal set of operations from which other, more convenient to use operations can be built. Practical relational database systems implement a number of other operators which we have not described here.

2.4 Constraints in the relation data model

A database for a particular application is designed by choosing a set of relation types that represent the entities and relationships in the application. This collection of relation types is called the database schema. The details of the mathematics of the relation model place

a number of constraints on the relation types in the database schema. A database schema that satisfies these constraints is said to be in normal form and the process of reshaping a candidate database schema design to meet the requirements of normal form is called normalization. The net effect of normalization is typically to scatter the attributes of an entity across many different tables.

The constraints of normal form are organized into various stages, first normal form, second normal form etc. In this context, we need only deal with first normal form.

First normal form requires that each column in a table must contain so-called "atomic" data. That is, the domain associated with the column must be some predefined, preferably fixed size type like an integer. The reason for this is that the relational operations see only the table structure and cannot deal with any internal structure associated with the data within a given cell in the table.

The most infamous type of non-atomic data is the array. Frequently, the most natural interpretation the application entity is that it has an attribute which is a variable length collection. For instance, a natural attribute for an employee might be "skills", a variable length array of skill keywords. However, this attribute would constitute a non-atomic attribute and hence is forbidden. Typically, the atomic attribute requirement forces the creation of additional tables, possibly EMPLOYEE_SKILLS in this case, which would cross reference employee entities to skill entities. In many applications this is an entirely acceptable approach but sometimes, as we'll see shortly, this is completely infeasible.

2.5 Impact of the relational model

The relational data model was a radical departure from previous data management approaches in that it was a *mathematical* model. Previous ad hoc approaches had mostly focused on how data was to be stored and described how to access the data in terms of how it was stored. This limited the types of queries that could be made and generated massive software maintenance problems whenever the data storage was reorganized.

The relational data model instead described data in terms of abstract mathematical objects and operations. The mathematical abstraction separated how data was accessed from how it was actually stored. Furthermore, the mathematics ensured that the relational algebra was a *complete* set of query operators, any query within the universe of possible queries defined by the model could be generated by a suitable combination of the fundamental relational algebra operators.

The mathematical abstraction and completeness of the relational algebra meant that sophisticated query processors could be implemented as independent subsystems, without knowledge of the application. This arguably created the database management system as a commercial product and unquestionably revolutionized the database industry.

3 Limitations of the relational model and the need for other models

In spite of the overwhelming success of the relational data model, not all application areas are well served by the model. In this section we describe three categories of applications which are not well served by the relational model and we describe the attempts to develop data models that match the requirements of these applications.

3.1 Spatial data

There are a wide variety of applications that deal with data that is spatial or geometric in nature. Computer aided design and manufacturing (CAD/CAM) and geographic information systems (GIS) are two well known, commercially important examples.

A main focus of systems that deal with spatial data is the need to represent spatial decomposition. Design data is organized into systems, subsystems, and parts. Geographical data is organized into states, counties, and cities. Furthermore, these applications frequently exhibit multiple, concurrent decompositions. For instance, geographic systems must represent both natural, physical boundaries and political boundaries.

At finest level of decomposition, spatial data consists of collections of geometric primitives and the topological relationships between the primitives. Geometric primitives include simple geometric shapes like points, lines, and polygons, as well as a wide and constantly growing zoo of mathematically more sophisticated primitives such as non-rational-uniform-B-splines (NURBS). The topological relationships describe the way these geometric patches are connected to form complex structures.

It has long been understood that the relational model is a poor choice for representing spatial data. There are (at least!) two fundamental issues:

- It is difficult to represent the decomposition relationships, especially the topological relationships, in a natural and efficient way. For instance, a polygon HAS_A collection of edges which would be naturally represented as an attribute of the polygon entity. However, first normal form prohibits such variable length collections as attributes. On the other hand, representing the topological relationships in separate relationship tables means it requires complex, possibly recursive, and frequently inefficient queries to retrieve all the parts of a geometric primitive.

- The operations of the relational algebra are not well suited to natural spatial queries such as nearness queries and region queries.

3.1.1 spatial data models

In response to these issues, there have been many attempts to extend the relational model to support spatial data, or to use other data models to support spatial data. Spatial data management is currently an active research topic.

3.2 Object-oriented programming

In the decade following 1985, there was a strong shift in software development practice from the older procedural programming style associated with languages such as FORTRAN, COBOL, and C to the object-oriented paradigm associated with languages such as Smalltalk, C++, and more recently Java.

The features of the object-oriented paradigm relevant to this discussion can be described using the entity-relationship notions introduced above. Object-oriented languages facilitate the definition of programmer-defined entity types called classes. Individual entities of these entity types are called objects. Complex entities and entity types are composed primarily using two relationships:

- The HAS_A relationship is used to compose simpler objects into more complex objects. That is, objects have parts which are other objects.

- The IS_A relationship is used to combine entity types into more complex types.

The IS_A relationship, or inheritance as it is called in the object-oriented paradigm, is a powerful new technique introduced by the object-oriented paradigm. The IS_A relationship is a relationship between entity *types*, rather than just individual entities. If entity type MANAGER is specified to inherit type EMPLOYEE, then the MANAGER type IS_A special kind of EMPLOYEE. Every MANAGER entity has all the attributes that that every EMPLOYEE entity has, plus any attributes that are specified in type MANAGER. This programming mechanism greatly facilitated the construction of complex software applications by making it much less labor intensive, and less error prone, to model the natural inheritance relationships found in applications.

In execution, an object-oriented application is a complex network of objects related by the HAS_A and IS_A relationships. The natural notion of data storage for such a system is the notion of object persistence. One should be able to easily store an object and all the objects it refers to in a database, thus making the object persist after the program that created it has finished execution. Similarly, one should be able to retrieve the object easily when execution resumes.

Attempts to use the relational model to store object-oriented data suffer one of the same difficulties described above for spatial data: complex, recursive HAS_A relationships are difficult to implement in the relational model.

An even more severe problem is that the IS_A relationship cannot be implemented directly in the relational model at all. In the context of a relational data base, the IS_A relationship is a relationship between relation *types*. As we saw above, a relation type is not a relation, it is just a set of attributes. Thus the relation types as such cannot be represented or operated on within the model.

3.2.1 The object-oriented data model

This fact has spurred the development an entirely different data model, the object data model, and its corresponding object-oriented database management system (OODBMS) technology. The object data model supports complex object composition (HAS_A relationships), inheritance (IS_A relationships), and object persistence in the context of some general purpose programming language like C++.

Although references to the "the" object data model are very common, there really isn't an object data model in a sense comparable to the relational model. The object model is essentially a family of related, ad hoc models, without any substantial industry or academic consensus on what "the" model is. The object model has no firm theoretical, mathematical basis and there is no set of operations equivalent to the relational algebra on which to base a query language. As a result, the definition of a general query language remains a problem to this day and OODBMS's tend to be tightly coupled to some general purpose object-oriented programming language (e.g. C++). OODBMS technology is widely perceived as obsolescent, being replaced by object-relational technology.

3.2.2 The object-relational data model

The object-relational data model is the second track of development motivated by the object-oriented paradigm. The object-relation model attempts to extend the relational model with oriented-oriented concepts derived from the relational formalism itself. The first normal form constraint is relaxed by allowing attributes that are themselves instances of relations. The object-relational approach retains a mathematical foundation and hence supports high-level query languages, extensions of the SQL standard developed from the relational model.

3.3 Numerical simulation and scientific computing

A third application area for which the relational model is not well suited, and an increasingly commercially important one, is numerical simulation or "scientific computing". Simulation software is aimed at predicting the outcome of complex physical, biological, financial, or other processes by building mathematical models and numerically solving the resulting equations. Defense, petroleum exploration, and medical imaging have been the classical applications for scientific computing. However, as the price of numerical computation has dropped, it has become increasingly cost effective to use simulation in a wide range of applications. For instance, it is an accelerating trend in manufacturing to replace the conventional design-build-test-redesign product development cycle with the so-called "virtual test" or design-simulate-redesign cycle. Similarly, financial trading is directed by market simulations and major metropolitan TV stations produce their own weather simulations, complete with computer generated animations.

Simulations combine features of spatial data and object-oriented data. The results of the simulation usually represent what is refered to in physics as a "field", that is, the dependence of some property on space or time. For instance, the result may represent the

dependence of mechanical stress on position within the product, or stock price on time, or temperature on location. Simulation data thus usually contains embedded spatial data representing the shape of the product, the interesting intervals of time, or the geography of the region of interest. In addition, the space and time dependent properties being computed usually are complex mathematical types with important IS_A relationships between them.

In addition to sharing these features with spatial data and object-oriented data, simulation data has another essential feature: the data sets tend to be very large. The amount of data that must be processed in a simulation is directly proportional to the desired accuracy. The quest for accuracy always requires that the simulations be run at (or over!) the limits of the computational resource.

As with the previous examples, the functionality provided by the relational model does not provide a good fit to the requirements of the application domain. The poor fit has historically led the scientific programming community to develop their own data models and supporting software.

3.3.1 Ad hoc scientific data models

The majority of scientific data models have been ad hoc models derived from assumed file formats. These are not true explicitly defined data models in the sense described above. Instead, the data format and file system has been implemented first, and the data model implied by the resulting functionality. Three of the most successful have been CDF ("Common Data Format") developed at NASA and its immediate descendant netCDF, develop at the National Center for Atmospheric Research, and HDF ("Hierarchical Data Format"), developed at the National Center for Supercomputing Applications at the University of Illinois.

In additional to these three relatively widely used models, there have been dozens of others, used within smaller communities or individual organizations. In particular, each of the national labs has developed several such systems.

3.3.2 The fiber bundle data model

In 1989 Butler and Pendley published the fiber bundle data model, the first true mathematical data model for simulation data. A succinct summary of the fiber bundle model is that the mathematical objects are generalized fields called sections of fiber bundles and the operations correspond to field algebra, calculus, and visualization.

There have been several implementations, or partial implementations, of the fiber bundle data model. Under contract to Sandia National Labs, LPS implemented the first prototype, the ViMS system, and a later more advanced version, the Whitney system. The best known implementation, and the only commercial one, is the IBM Data Explorer system.

Experience with these implementations, and especially the extensive effort starting in 1998 within the Data Models and Formats project of the Advanced Strategic Computing Initiative ("ASCI") funded by the U.S. Department of Energy has exposed critical limitations of the fiber bundle model. Specifically, it became clear that although the fiber bundle model does a good job of describing the interface to field data, that is, how we want to use the data, it does not deal adequately with describing the representation of the data, that is, how we store the data. In short, the fiber bundle data model describes the field abstraction, but doesn't describe the data!

4 The sheaf data model

The sheaf data model is a generalization of the relational data model. Historically, the sheaf data model evolved out of the fiber bundle data model in order to remove the observed limitations and to provide additional features required by ASCI applications. In the current presentation however, we will present the sheaf data model as a generalization of the relational data model, in order to clarify the difference between the two models.

4.1 The importance of inclusion relationships

The essential insight that leads from the relational model to the sheaf model is to observe that the difficulties using the relational model to represent spatial, object-oriented and scientific data all originate in the fact the relational model provides no explicit mechanism for representing *inclusion*. There are two distinct types of inclusion we need to represent:

row inclusion: HAS_A relationships, e.g. the decomposition relationships of spatial data and the object containment relationships of object-oriented data, correspond to row inclusion. For instance an edge entity, represented by a row, is conceptually included in a polygon entity represented by another row.

column inclusion: IS_A relationships, e.g. the inheritance relationship of object-oriented data, corresponds to column inclusion. If type MANAGER inherits type EMPLOYEE, then a MANAGER table includes all the column headings of an EMPLOYEE table.

What is needed is a generalization of the relational model that incorporates these two types of inclusion.

4.2 The mathematics of inclusion: partially ordered sets

The mathematicians have developed an extensive theory of inclusion known as the theory of partially ordered sets. A partially ordered set, or poset as it is frequently abbreviated, consists of two components: a set of objects, called the base set, and a relation on the base set, called the ordering relation. The ordering relation explicitly *defines* which members of the base set are included in each other.

We can extend the table analogy we used for relations to describe a partially ordered set. In the table picture, a poset consists of a table and a graph. The table represents the base set and the graph represents the ordering relation. A graph is a diagram that consists of

points or "nodes" connected by lines or "links". An example will help make this picture clearer. Figure 11 shows a very simple spatial object: a single triangle consisting of three vertices, three edges, and the triangle itself, including the space in the interior of the triangle.

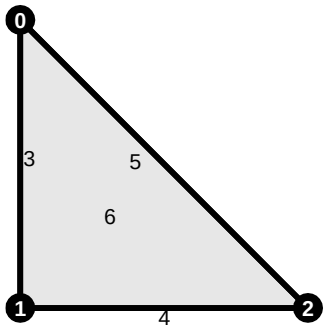
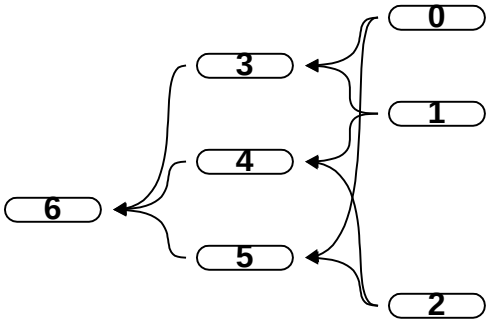


Figure 11: A very simple spatial object, labeled with entity IDs

Figure 12 shows the table picture of the corresponding poset. The table represents the base set, with one row for each of the entities in the triangle: the three vertices, the three edges, and the triangle itself. The graph at the left hand side represents the ordering relation, that is the inclusion relationships. Each "node" in the graph is labeled with the ID of the entity it represents and there is a "link" from each node to the node representing the *next most inclusive* entity. (Note that the links are directional.) An entity is included in another entity if there is a path in the graph from the node corresponding to the first entity to the node corresponding to the second entity. So, for instance, by examining the graph we can see that vertex v0 (ID = 0) is included in the triangle t0 (ID = 6) because there is a link from node 0 to node 3 and a link from node 3 to node 6.



SIMPLE_SPATIAL_OBJECT		
ID	NAME	TYPE
0	v0	vertex
3	e0	edge
1	v1	vertex
4	e1	edge
6	t0	triangle
5	e2	edge
2	v2	vertex

Figure 12:The SIMPLE_SPATIAL_OBJ poset.

If entity A is "next most inclusive" to entity B, it simply means there is no entity C that includes B but is included in A. Thus for any given entity, there may be more than one next most inclusive entity. For instance both edge e0 and edge e2 are "next most inclusive" to vertex v0. This is exactly the meaning of the term "partial order"; the members of the base set are ordered with respect to each other, but not totally ordered so that every member has exactly one "next larger" member. In a partial order, a member can have one next larger, many next largers, or no next largers. Indeed, the definition of a partially ordered set includes the case in which every member of the base set has no next larger members. This corresponds the ordinary, unordered set.

The ordering relation of a partially ordered set is a generalization of the familiar relation "less than or equal to". It is customary to call the ordering relation by that name and to refer to it with the symbol "≤". So, for instance, we can say v0 ≤ e0 in the above example, meaning that v0 is included in e0.

4.3 Generalization of the relational algebra operators

All the operators of the relational algebra can be generalized to work on posets. The effect of any of the operators on the table part of a poset is the same as in the relational case, but each operator must be generalized to also operate on the graph part. We will revisit this generalization in more detail, once we have introduced some additional concepts.

4.4 The poset algebra operators

In addition to the relational algebra operators, there are a number of operators that are derived from the order relation. We describe two of the most important here. In the following, a_1, a_2, etc. are rows in a poset table A.

4.4.1 is_less_than_or_equal_to or "≤"

a_1 is_less_than_or_equal_to a_2, if and only if there is a path in the graph from the node representing a_1 to the node representing a_2.

4.4.2 down

Down(a) returns a poset that contains all the members of A that are less than or equal to a. The result is called the down set of a. Figure 13 shows the result of the down operator applied to the member with id = 4 in the SIMPLE_SPATIAL_OBJECT poset.

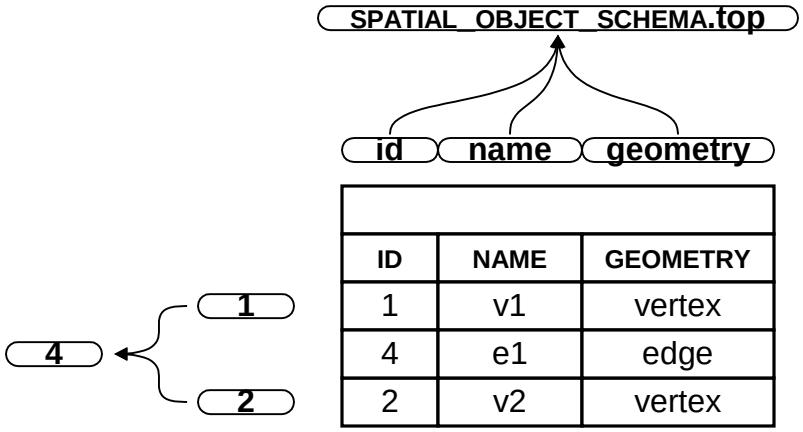


Figure 13: down(SIMPLE_SPATIAL_OBJECT.ID = 4)

The name "down set" originates in the standard mathematical practice of drawing the links such that if $a_1 \leq a_2$, then a_1 is below a_2 in the graph; smaller members are always below larger members. In such a "vertical" graph, the down operator literally goes down the graph from its argument, hence the name. In order to be visually consistent with the rows in the table, we have drawn our graph so that the links always point from right to left. Perhaps we should call it the "right" operator, given the way we draw the graph!

4.5 Lattices and the lattice algebra operators

The operators we have defined so far operate on the entities that specifically appear in the table. In the applications in which inclusion plays an important role, it is often important to treat a collection of entities as if it was an entity itself, a sort of "virtual" row in the table. For instance, in spatial applications it is often important to deal with the boundary of an object. The boundary of our SIMPLE_SPATIAL_OBJECT is a collection of three edges, but we would like to treat it as a single entity, the boundary.

Lattice theory, another branch of the mathematical theory of inclusion, allows us to do exactly that. Associated with every finite poset is another poset, a special kind of poset called a finite distributive lattice ("FDL"). Figure 14 shows the FDL for the SIMPLE_SPATIAL_OBJECT poset. The number of members of an FDL lattice is always much greater than the number of members of the poset that generates it. For simplicity and readability, we have drawn only the graph, in the traditional mathematical orientation. We have also substituted icons showing the geometrical meaning of each member of the lattice instead of ID attributes as used in previous figures.

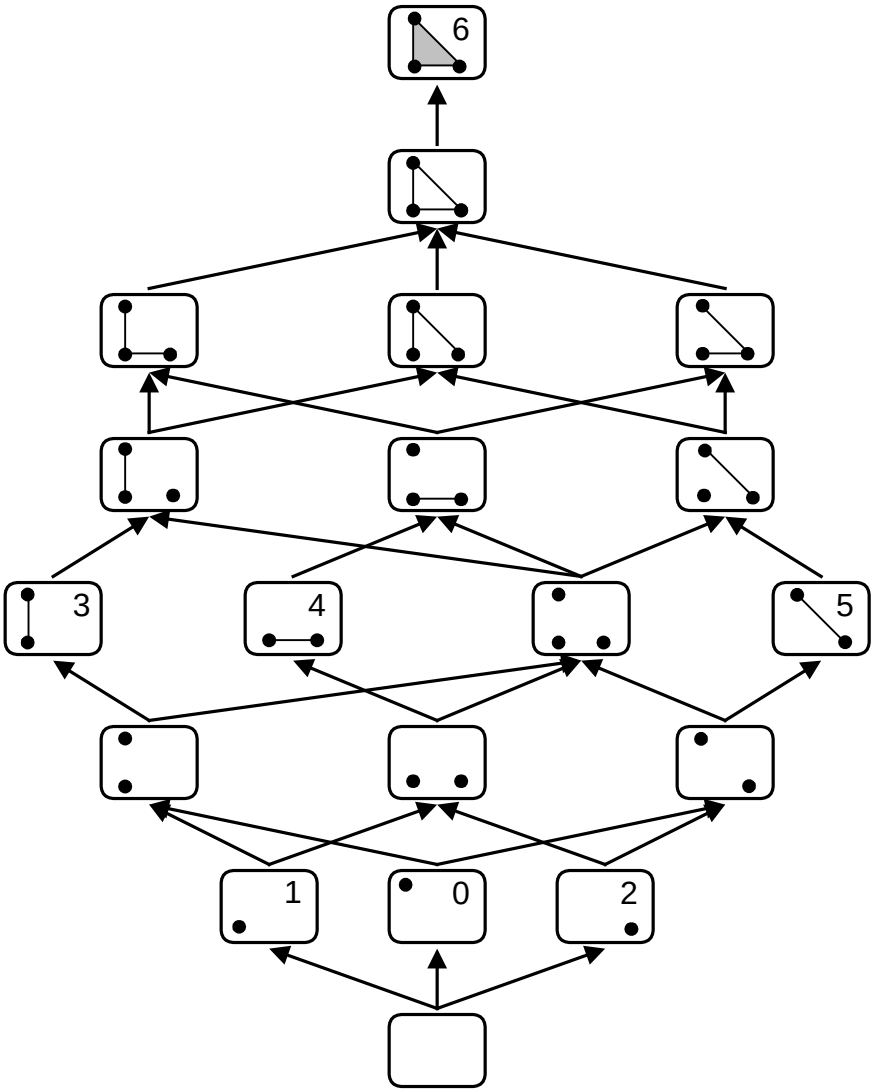


Figure 14: The finite distributive lattice for poset SIMPLE_SPATIAL_OBJECT.

Intuitively, the lattice contains all the members of the original poset plus all distinct combinations of them. Two different combinations of members are distinct if they do not represent the same inclusion. For instance, the combination of just edges e0 and e1 is distinct from the combination of edges e0, e1, and e2, since the former combination does not include all the points in edge e2. Conversely, the combination of the three edges is not distinct from the combination of the three edges and the three vertices. These two combinations (edges and edges+vertices) are equivalent because the vertices are already included in the edges; adding them to the combination doesn't include any additional points.

As consequence, the lattice in Figure 14 contains a single member for the boundary, it does not contain separate members for the collection of three edges and for the collection

containing three edges and three vertices. The lattice is *not* the set of all subsets of rows of the table. The formal mathematical definition of the lattice is that it is the set of all *down sets* of the original poset, but we won't need to go into that. The notion of all distinct combinations will do.

So the lattice generated by a poset is a poset itself and can be thought of in the table + graph picture. Its table contains all distinct combinations of rows of the original poset and its graph contains all the inclusion relations.

In a lattice there are two additional operations defined. In the following l_1, l_2, … l_n are members of a lattice L.

4.5.1 Join or least_upper_bound

Join(l_1, l_2, … , l_n) returns unique the smallest member of L which is greater than or equal to all of its inputs.

Note: the term join has another meaning in the context of relational data bases, but since we will not be using the term in its relational meaning, we hope no confusion will result.

4.5.2 Meet or greatest_lower_bound

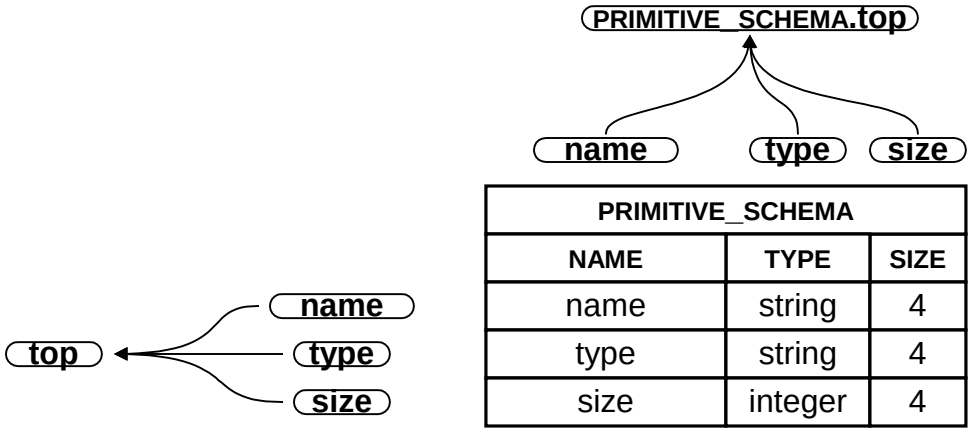
Meet(l_1, l_2, … , l_n) returns unique the largest member of L which is less than or equal to all of its inputs.

4.6 The sheaf operators

So far, we have dealt only with row inclusion. We still need to incorporate column inclusion into the model. To do this, we need to expand our table + graph picture so that there are two graphs associated with the table. The graph we already had, which we will now refer to as the row graph, and a new graph, which we will call the column graph. The column graph, naturally enough, describes the inclusion structure of the columns.

In the sheaf data model, every lattice has an associated lattice, called its schema. The column graph of a lattice is defined by the row graph of its schema. The schema relationship is recursive: a schema lattice has to have a schema lattice. This recursion is terminated in a special lattice, the primitive schema lattice, which is its own schema.

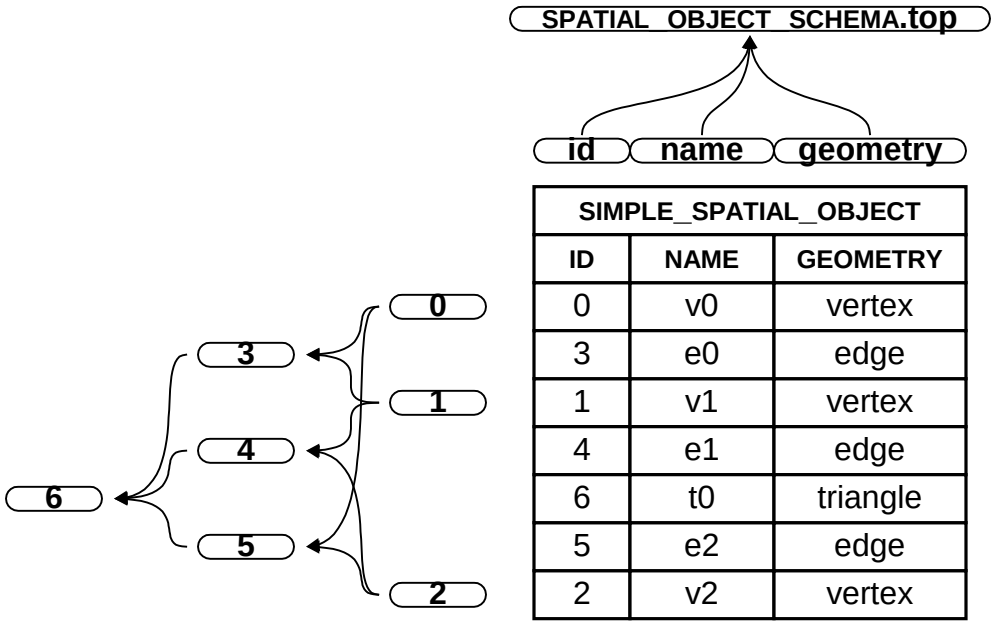
A simple primitive schema lattice is shown in Figure 15. Note that the primitive schema lattice uses its own row graph as its column graph.



PRIMITIVE_SCHEMA		
NAME	TYPE	SIZE
name	string	4
type	string	4
size	integer	4

Figure 15: A simple primitive schema lattice.

The previous SIMPLE_SPATIAL_OBJECT lattice with its corresponding schema lattice is shown in Figure 16 and Figure 17, respectively.



SIMPLE_SPATIAL_OBJECT		
ID	NAME	GEOMETRY
0	v0	vertex
3	e0	edge
1	v1	vertex
4	e1	edge
6	t0	triangle
5	e2	edge
2	v2	vertex

Figure 16: The row and column graphs for SIMPLE_SPATIAL_OBJECT

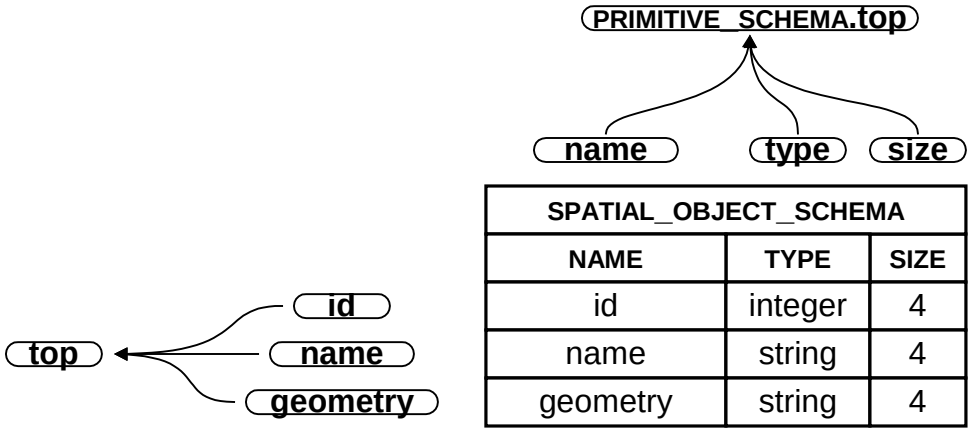


Figure 17: The schema lattice for SIMPLE_SPATIAL_OBJECT

The association between a lattice and its schema lattice introduces three more operators:

4.6.1 The exponentiation operator

The exponentiation operator *EXP*(L), returns a lattice with schema L. The table and row graph of the result are empty.

4.6.2 The logarithm operator

The logarithm operator, *LOG*(L), returns the schema lattice associated with lattice L.

4.6.3 The restriction operator

RESTRICT L *TO* s, where s is a member of the schema of L, returns the projection of L onto the columns in the down set of s.

4.6.4 Sheaves

The table of a lattice L is the Cartesian product of the rows of the schema lattice, where each row is interpreted as a domain. By using *RESTRICT* L *TO* s successively for each member s of the schema lattice we can define a family of tables, one for each member of the schema. Each member of this family is itself a Cartesian product over a subset of the rows of the schema lattice.

An association, or map, generated in this manner between a schema lattice and a family of Cartesian product sets is called a sheaf. The schema lattice is referred to as the source of the sheaf and the family of product sets is referred to as the target of the sheaf. We will call the largest member of the target, the table which is restricted to produce all the other members of the target, the top table.

The row graph of the top table is not defined by the schema. It can be arbitrarily chosen by the user. Once the row graph is defined, the top table defines an FDL and can itself be used as the schema for another lattice.

Hence every lattice defines a sheaf of Cartesian product sets (tables) and this sheaf can be viewed as the primary object of the data model. The table, row graph, and column graph, as well as the relational, poset, lattice, and sheaf operators can all be considered different aspects of the sheaf object. This is the origin of the name sheaf data model.

4.6.5 Data dependent schema

The sheaf structure and operators introduce column inclusion into the data model, but they also an additional, critical feature: data dependent schema. The schema of a table is defined by the contents of another table, not predefined to some fixed set of columns. This feature is essential for representing simulation data.

4.7 Generalization of the relational algebra operators revisited

We have now introduced all the additional concepts we need and can return to the relational operators. As we mentioned above, all the operators of the relational algebra can be generalized to work on sheaves. The effect of any of the operators on the table part of a sheaf is the same as in the relational case, but each operator must be generalized to also operate on the row and column graphs. We quickly sketch the appropriate generalization for each of the six operators.

4.7.1 Cartesian product

The column graph (schema) of the Cartesian product $A \times B$ of two sheaves is the disjoint union of the column graphs of A and B. This means that, as in the relational model, the table of the product has the columns of A and the columns of B, with rows for each possible pair of values, one from A and one from B.

However, there are several possible choices for the row graph part. Let (a_1, b_1) and (a_2, b_2) be two rows in the product table:

- product order: $(a_1, b_1) \leq (a_2, b_2)$ precisely when $a_1 \leq a_2$ and $b_1 \leq b_2$
- lexicographic order: $(a_1, b_1) \leq (a_2, b_2)$ precisely when $a_1 \leq a_2$ or $a_1 = a_2$ and $b_1 \leq b_2$
- user-defined order: any order relation the user wishes to define

The choice of order must be provided as a parameter to the operator.

4.7.2 selection

The table part of the result is the same as for the relational operator. The column graph of the result is the same as the column graph of the input. The row graph is the same as the input, but all nodes and links that refer to rows that aren't in the result are removed.

4.7.3 projection

The column graph of the result is the column graph of the input, with all nodes and links that refer to columns that are not in the output removed. The row graph of the result is the same as the row graph of the input, except that if any rows are discarded because they are duplicates, then so are the nodes and links in the graph that refer to them.

4.7.4 union

The column graph of the result is the same as column graphs of the two inputs, which must be the same as each other. The table part is the same as in the relational model. The row graph is the union of the row graphs of the input posets.

4.7.5 intersection

The column graph of the result is the same as the column graphs of the inputs. The table part is the same as in the relational model. The row graph is the union of the row graphs of the input posets, with all nodes and links that refer to rows that aren't in the result removed.

4.7.6 rename

The rename operator is the same as in the relational model.

4.8 Summary

The sheaf data model provides a formal, mathematical data model that generalizes the relational data model and explicitly incorporates inclusion. The algebra associated with the model contains relational, poset, lattice, and sheaf operators which provide the basis for a complete data definition and manipulation language.

5 Implementation of the sheaf data model

The sheaf structure and its operators form an extremely useful abstraction. It suffers from a severe practical problem however. Remember that the source of a sheaf is the finite distributive lattice (FDL) containing "all possible distinct combinations" of members of a client-defined poset. The problem is that even for very small posets, the number of "all possible distinct combinations" of rows is *astronomical*. It is completely infeasible to generate and store the entire source lattice. For the same reason, it is infeasible to store the target of the sheaf, i.e. the family of product sets.

A major component in the cluster of insights and inventions described in this document is the combination of data structure and algorithms that allow us to utilize the sheaf formalism without actually generating the entire source and target. We refer to this combination as the finite sheaf data type. In order to understand this data type, we first have to introduce a few facts about FDLs.

5.1 Some features of finite distributive lattices

As we said above, an FDL contains the members of the original poset and all distinct combinations of them. The members of the original poset are referred to as the join-irreducible members or "jims" of the lattice, while the "distinct combinations" are "join reducible members" or "jrms" (pronounced "germs").

The central theorem of the theory of FDLs is the Birkhoff representation theorem, which states that a jrm is equal to the join of the collection of jims contained in its down set. (This collection of jims is the "distinct combination" associated with the jrm.)

From the usual mathematical point of view, an FDL is a given, pre-existing base set and ordering relation. We can think of the table and graph as fully instantiated. Every member of the lattice is represented by a row and a node in the graph; every inclusion relationship is represented by a path in the graph. From this point of view, the Birkhoff representation theorem is a statement about a relationship that must exist between the nodes and links of the graph. The join and meet operators are queries that find the proper member using the (fully instantiated) graph.

5.2 The basic representation strategy

The fundamental mechanism that makes it practical to use the sheaf formalism is a form of deferred computation or "lazy evaluation". Instead of mimicking the mathematics and instantiating the entire lattice, we instantiate the jims and only the jrms the user specifically requests. This approach reduces the storage requirements to feasible levels, but forces extension and reinterpretation of the mathematics.

5.3 The finite sheaf data type

The finite sheaf data type consists of a data structure and a collection of operators on the data structure that implement the lazy evaluation strategy described above.

5.3.1 Data structure

The data structure consists of the following components and interpretations:

- schema: a reference to another instance of a finite sheaf data type
- table: a collection of records. The table and record types are widely used in computer science and can be represented in a variety of ways, any of these well-known methods are suitable. There must be a one-to-one correspondence between rows in the schema and columns in the table.
- row graph: a directed acyclic graph. The directed acyclic graph ("DAG") is also a well-known data structure with a variety of representations, any of which are acceptable. There must be a node in the graph for each row in the table, but the graph in general will contain additional nodes, not associated with a specific row in the table.

The interpretation of this data structure is that it represents the top table of the sheaf defined by the schema. Each row in the table represents a jim in the row lattice of the table.

The row graph represents the order relation for the row lattice. Nodes in the graph which are correspond to rows in the table represent jims in the lattice, while the remaining nodes represent jrms. The links in the graph represent the "next most inclusive" relation (also known as the cover relation in mathematical lattice theory). The collection of jims associated with each jrm by the Birkhoff theorem is generated by traversing the graph below the jrm.

In the usual mathematical formulation of an FDL, each member is unique. But in practical computing applications, a user may want to have multiple copies of a lattice member. The data structure supports this by having the graph represent a "lexicographic" ordering relation. A lexicographic ordering relation is a generalization of the order words appear in a dictionary. Words are first ordered by their first letter. All words with the same first letter are then ordered by their second letter, etc. The "first letter" in the finite sheaf order is derived from the Birkhoff theorem. Member l_1 is less than member l_2 if the set of jims in the down set of l_1 is a subset of the set of jims in the down set of l_2. The "second letter" is the order the members were created in.

Lattice members which are copies have the same set of jims in their down set, the same "first letter", and are ordered relative to each other by the "second letter", the order they were created in.

The table and graph combination stores all the jims of the row lattice, but only those jrms that the user specifically creates.

5.3.2 Operators

The finite sheaf data type supports all the relational, poset, lattice, and sheaf operators of the sheaf data model, as described above. In addition, the finite sheaf data type extends and reinterprets the mathematics as follows:

First, we must extend the set of operators to include operations for creating the jims and the order relation:

CREATE_JIM: creates a jim as a row in the table and corresponding node in the graph

DELETE_JIM(JIM_ID): deletes the row and node corresponding to the jim with identifier jim_id.

CREATE_LINK(LESSER_ID, GREATER_ID): creates a link in the graph between the jims identified by lesser_id and greater_id

DELETE_LINK(LESSER_ID, GREATER_ID): deletes the link between the jims identified by lesser_id and greater_id.

Second, we must reinterpret the join and meet operators. Mathematically the result of these operations is guaranteed to exist, and hence these are query operations. They find the proper member using the order relation (i.e. row graph.) But if only previously requested jrms have been instantiated, then the result of a meet or join may not exist. The result has to be created and linked into the graph. In other words, instead of deriving the result from the order relation, the order relation must be derived from the result.

5.4 Efficient access to secondary storage

Problems of practical interest require the table and graphs associated with the finite sheaf type to be stored on disk, outside of the main memory of the computer. Efficient access to such externally stored data must take into account the properties and performance characteristics of disk storage. This problem has been extensively studied and solved in the context of relational data base management technology.

The standard solution relies on constructing hierarchical index structures which allow retrieval of any record with a minimum number of disk accesses. The best known index structures are the so-called "B-tree" and it variants. Nodes in the B-tree correspond to hierarchical groupings of records in the table.

The key to efficient indexing of the finite sheaf is the following observation: *the hierarchical groupings of the B-tree are jrms in the row lattice*. They are jrms chosen specifically to optimize disk access. Hence, *the row graph itself can be used as an index.* In addition to the jrms explicitly constructed by the user, internal routines of the finite sheaf type can construct jrms intended purely for use in achieving efficient disk access.

6 Applications

6.1 Spatial data

The mathematical study of spatial structure is organized into two broad disciplines. Topology is the study of continuity, nearness, and connectivity without regard to explicit shape and size. Geometry adds shape, size and measure to topology. It is a well-established, but not widely known, fact of mathematics that the theory of topology can be formulated entirely in terms of lattice theory. Furthermore, recent research in computational geometry has established that all existing methods of geometry representation can be described using finite distributive lattices.

The sheaf data model provides a direct realization of this mathematical structure and hence is ideal for storing and manipulating spatial data.

There is however, an additional operator, not described above, that is particularly convenient for creating instances of the finite sheaf type representing spatial data. Spatial data often appears in the form of a mesh, a large number of interconnected geometric primitives, all the same type, or of related types. As a simple example, the shape of a product may be specified in a CAD/CAM system by a triangle mesh - a large number of

triangles connected together at their edges so as to cover the surface of the product. Such meshes are even more common when spatial data is used in the context of a simulation.

Mesh data is usually presented by specifying the number and type of the geometric primitive and by specifying the so-called "connectivity" data. The connectivity data describes how the primitives are to be connected, typically by listing the vertices that appear in each primitive. So a triangle mesh could be specified by stating the number of triangles, then specifying 3 vertex identifiers for each triangle. Two triangles are implicitly connected at a point if they both contain the same vertex, and connected at an edge if they share two vertices.

The additional operator is the cellular lattice constructor that makes it convenient and efficient to build the table and graph of the lattice corresponding to a mesh, given the usual input data.

6.2 Object-oriented data

Recent research in computer science has demonstrated that the inheritance relationships between classes in an object-oriented application generate a mathematical lattice. The sheaf data model is thus able to directly and precisely represent inheritance relationships.

Object containment relationships can be divided into two categories: cyclic and acyclic relationships. In a cyclic relationship A contains B contains C contains … contains A. In other words, the chain of relationships eventually forms a circle. In an acyclic relationship, the chain is linear. The sheaf data model, as stated, can represent only acyclic containment relationships.

The combination of strong support for IS_A relationships with support for acyclic HAS_A relationships makes the sheaf model a good, but not ideal, fit for object-oriented data.

6.3 Numerical simulation and scientific data

As we described above, simulation results typically represent the dependence of some property on space and time. In mathematical physics, such space and time dependent properties are called fields. Once again, it is a well-established but not well known fact that the abstract fields of mathematical physics can be represented by sheaves.

An important part of the invention described here is the method by which the concrete field data of numerical simulation can be interpreted as a sheaf.

An field is a map that associates a value of some property, called the fiber space, with every point in some object, called the base space. Both the fiber space and the base space can be represented as finite sheaves. The representation of an abstract field as a finite sheaf requires the following information be provided:

- Base space sheaf, for instance Figure 16

- Fiber space schema sheaf, for instance Figure 18

- Discretization map: a map that associates each member of the base space lattice with a member of an arbitrary finite sheaf called the discretization of the base space. For instance, Figure 19 shows a discretization map and Figure 20 shows the corresponding discretization. The image of a member of the base space lattice under the discretization map is also called the discretization of the member. Typically the discretization is generated from the base space lattice itself. For instance the jims of the discretization are chosen to be the set of all the vertices in a mesh and the discretization jrm associated with a base space jrm is the subset of vertices contained in the down set of the base space jrm.

- Evaluation subposet: a subset of the base space lattice. This subset must be chosen so that it covers the base space, for instance, the subposet containing only member 6 (the entire triangle) in Figure 16. (In a larger example, a typical evaluation subposet would be the set of all triangles in a triangle mesh.) Every member of the evaluation subposet is assumed to carry a local coordinate system.

- Evaluation method: a rule that can used to compute the value of the field given the local coordinates of a point in a member of the evaluation subposet and the degrees of freedom (defined below) associated with that member. For instance, a common evaluation method on the triangle would be linear interpolation.

The schema, Figure 21, for the sheaf of fields, Figure 22, with a given base space and fiber space is the tensor product lattice of the discretization and the fiber space schema. The jims of the tensor product lattice are all pairs of the form (d, f) where d is a jim of the discretization and f is a jim of the fiber space schema. As with any finite sheaf, there is a column in the table of the sheaf for each jim in its schema. The domain of the column associated with schema jim (d,f) is defined to be the domain of f; the domain of d is ignored. A field is represented by a row in the table and the data in the cells of the row is referred to as the degrees of freedom of the field. The degrees of freedom associated with any pair (b, f), where b is a member of the base space lattice and f is a member of the fiber schema, is the restriction of the row to the schema member (d(b), f), where d(b) is the discretization of b.

Any numerical representation of a field can be interpreted as a sheaf using the above method. The ability of the sheaf data model to directly represent arbitrary fields, in addition to traditional relational data, spatial data, and object-oriented data, makes it an ideal model for scientific data.

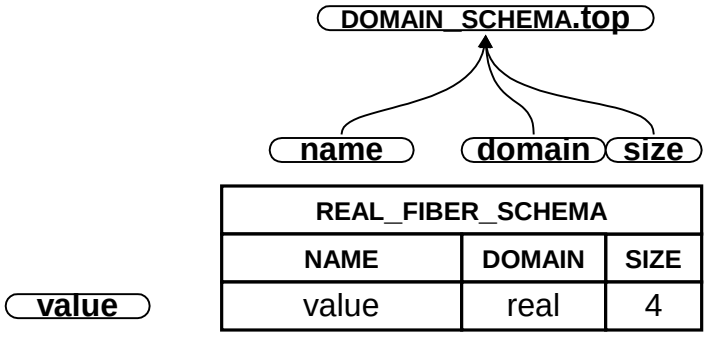


Figure 18: A fiber space schema for a real-valued field.

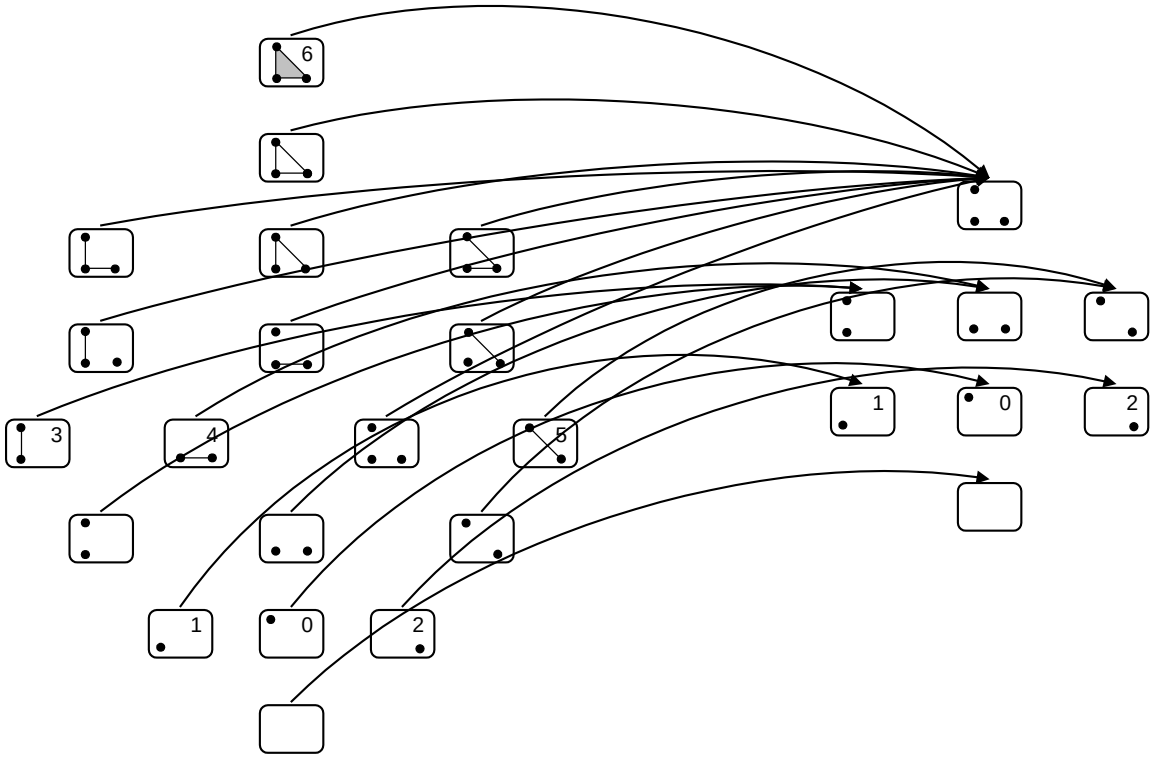


Figure 19: A discretization map for SIMPLE_SPATIAL_OBJECT

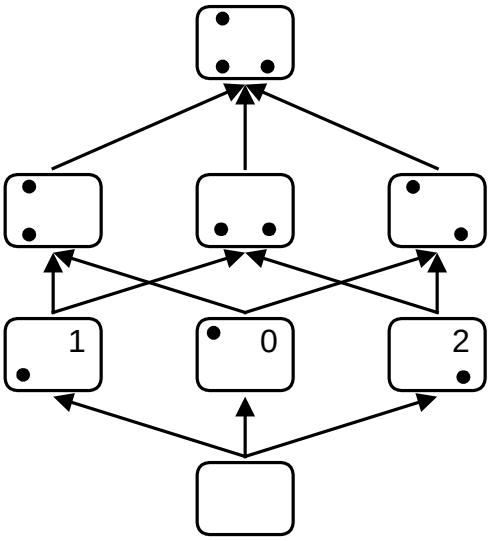
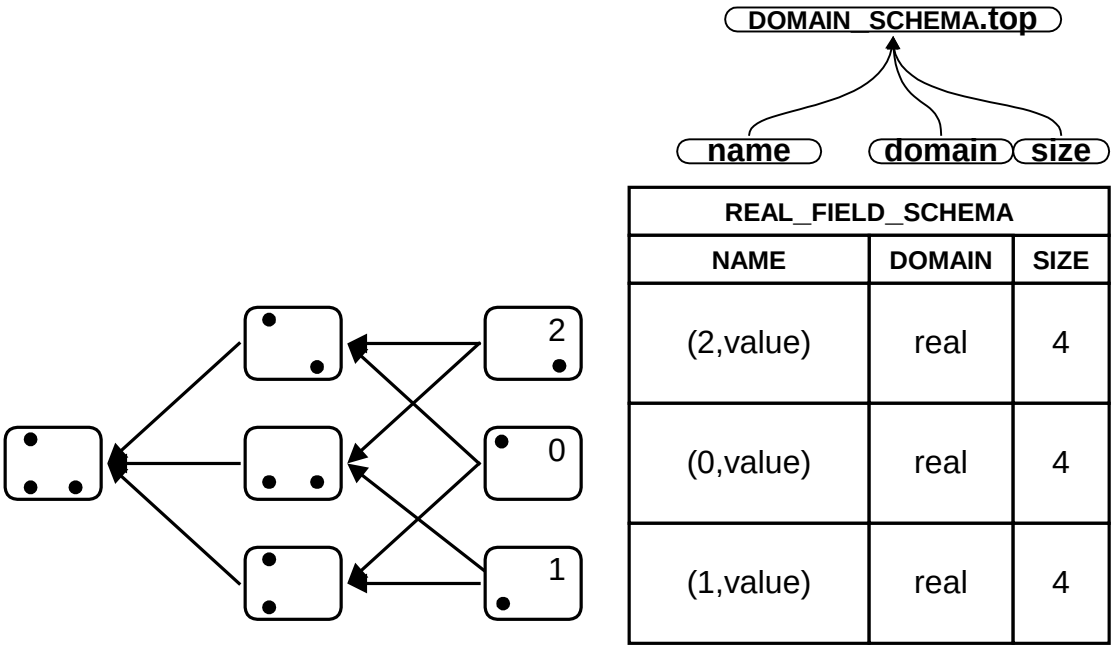
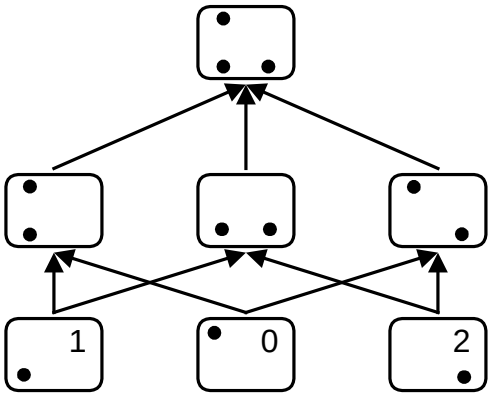


Figure 20: A discretization of SIMPLE_SPATIAL_OBJECT



REAL_FIELD_SCHEMA		
NAME	DOMAIN	SIZE
(2,value)	real	4
(0,value)	real	4
(1,value)	real	4

Figure 21: The schema for a real-valued field on SIMPLE_SPATIAL_OBJECT



	REAL FIELD ON SIMPLE SPATIAL OBJECT		
	(1,VALUE)	(0,VALUE)	(2,VALUE)
0	24.0	81.6	-10.2
1	230.5	3.4	98.7
2	-45.2	567.9	26.6
3	7.0	7.0	7.0

Figure 22: The sheaf REAL_VALUED_FIELD_ON_SIMPLE_SPATIAL_OBJECT

7 Release history

Release 1.0: 5/12/2000. Initial release for provisional patent application.

Release 1.0.49: 8/7/2011. Confidentiality statement replaced by copyright statement and reference to external vugraphs in fiber bundles discussion removed.

Release 1.2: 10/1/2012. Typos corrected. Release history added.