Sheaf System Analysis and Design Tutorial

David M. Butler

Limit Point Systems, Inc.

1 Introduction

The companion tutorials Part Space and Part Spaces for Scientific Computing gave a primarily non-mathematical introduction to the concepts of the sheaf data model. This tutorial provides the complement - it introduces the mathematical concepts and how they can be used in the analysis of problem domains and the design of software.

Full treatment of all the mathematical concepts we will discuss would be a major undertaking, well beyond the scope of this tutorial. Instead we describe the fundamental notion of each concept and usually the basic definitions. We then suggest additional reading with pointers into the references given in Appendix A.

We assume the reader has read the companion tutorials and is familiar with the concepts presented there. We also assume that the reader is familar with the fundamental ideas of mathematical set theory: set, subset, inclusion, intersection, and union.

2 A framework for domain analysis

Conventional software analysis and design is a process conducted within the scope of some specific product or application. The requirements statement for the application is the input to the process and a software design is the output. Domain analysis, by contrast, is a process based on the tenet that a greater degree of software reuse can be realized by broadening the scope of the analysis and design activity to a family of related applications, an application domain, rather than just a single product. The level of mathematical abstraction associated with the sheaf data model naturally applies to a broad scope of applications, nominally all of scientific computing, so it is natural to organize the concepts into a framework for domain analysis.

In order to introduce the framework without presupposing knowledge of the mathematical concepts we're going to put in it, we have to start with the simple notion of a "property association". We won't really define this notion, just give some examples:

- associate the speed of a car with every instant in a time interval,
- associate an annual income with each person in some group
- associate a temperature with every point in a room, and
- associate a comfort level with every temperature.

Such associations are ubiquitous in domains of practical interest and they have been studied extensively in mathematics. The mathematical theory identifies three roles within such associations:

- The entity that carries the property is called the base space. Every point of the base space gets at least one property value associated with it.
- The set of all possible property values is called the fiber space. Any given property value may or may not be associated with the base space.
- The association itself is called a section. The set of all possible associations between a given base space and fiber space is called a section space.

The motivation for these names will become clear later. In the mean time, we can identify these roles for each of the initial examples:

- Associate the speed of a car with every instant in a time interval: speed is the fiber space, the interval of time is the base space. The speed as a function of time is a section. The car might have proceded with a different speed profile, that would be another section.
- Associate an annual income with each person in some group of people: income is the fiber space, the group of people is the base space. The association of income and person in a given year is a section. In some other year the incomes might be different, which would be another section.
- Associate a temperature with every point in a room: temperature is the fiber space, the room is the base space. At any given time, there is a particular temperature associated with each point and this association is a section. At a different time the temperatures might be different, this would a different section.
- Associate a comfort level with every temperature: comfort level is the fiber space, temperature is the base space. Different people might assign different comfort levels, the comfort profile for each person would be a different section.

As the last two examples show, base space and fiber space are indeed roles, not types. A given space can play different roles in different associations.

Along with these three roles, our framework will be organized around three aspects:

- The conceptual aspect concerns the problem domain described in its own language, without regard to computer representation.
- The computational aspect concerns the entities of the conceptual aspect reformulated for computer representation, approximation of infinite spaces by finite spaces in particular.
- The data aspect concerns the entities of the computational aspect reformulated as persistent data, with no notion of procedure invocation or execution.

These three aspects can be roughly compared to three common phases of conventional analysis as shown in Figure 1. However, as suggested graphically, the three conventional phases are considered separate models while, as we shall see, the three aspects of the sheaf approach form a single, integrated mathematical model.

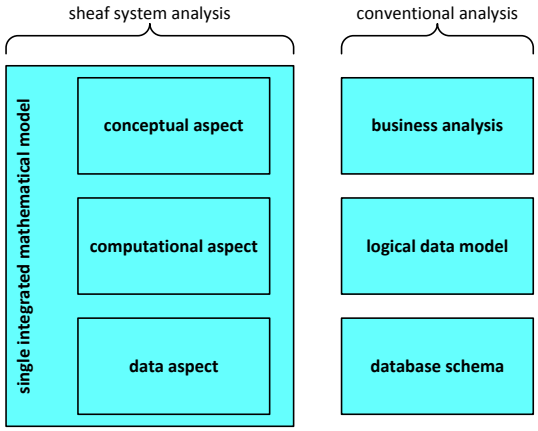


Figure 1: Sheaf system analysis aspects compared with conventional analysis.

The three roles and the three aspects form a matrix of analysis activities and the mathematical concepts of the sheaf data model can be viewed as a mathematical toolkit for carrying out the activities. The analysis activities for the framework and associated mathematical tools are shown in Figure 2.

This tutorial will describe the mathematical tools available for each of the activities, proceeding across the rows, starting with the conceptual aspect.

3 An extended example

We will introduce the mathematical concepts in the context of an extended example based on wells and well logs. We'll use a branched well with one side bore, a schematic representation of which is shown in Figure 3. The terms "upper well" and "lower well" are not standard terminology for wells, but the meaning should be clear enough and we will need to refer to these parts at various points in the discussion.

	base space	fiber space	section
conceptual aspect	Sets Topological spaces Poset of parts	Arbitrary sets Abstract vector spaces Affine spaces Transformation groups	Maps Fiber bundles Section space
computational aspect	Poset refinement Finite distributive lattice Cell decomposition	Classes Vector component rep Tensor class hierarchy	Parameterized computable functions Parameter spaces Local section space Global section space
data aspect	Lattice-ordered relations	Sheaf of lattice-ordered relations Lattice schema	Local section schema Global section schema

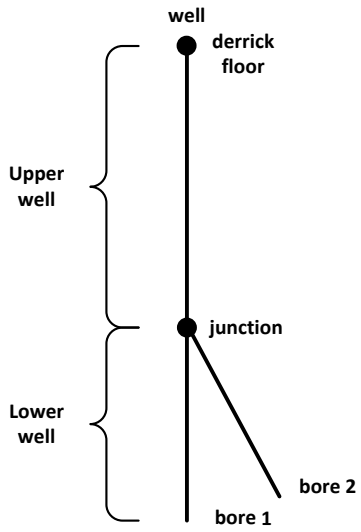


Figure 2: A framework for domain analysis.

4 Conceptual aspect for the base space role

We'll start our row-major traversal of Figure 2 in the upper left-hand corner, the conceptual aspect of the base space role. We consider our well as a base space, that is, something we can associate properties with. There are 3 primary tools in the mathematical toolkit for describing base spaces: point set, topological space, and partially ordered set. These tools form a layered set of abstractions, both topological space and partially ordered set add structure to the point set abstraction. For the reader familiar with object-oriented programming, it may be useful to think of this layering of abstractions as very similar to inheritance. A topological space is_a point set and a partially ordered set is_a point set.

4.1 Point set

Point set is the least structured, most abstract of these concepts. The fundamental notion of a point set is pretty simple: a point set is just an unstructured collection of featureless objects, as depicted graphically in Figure 4. By unstructured, we mean there is no notion of order, dimension, shape, or position associated with a point set. By featureless, we mean that, at least as far as point set theory is concerned, the objects have no properties except identity, we can somehow tell one from another, and we can tell whether a given object is in our collection or not. That is all; a point set is just a blob of points. In fact, it's not even a blob because blobiness suggests dimensionality.

The word "point" in "point set" is traditional, but it doesn't really mean much. A point set is just a set and a point is anything we want it to be. A point is specifically not required to be a coordinate tuple (x, y, z) as it is sometimes defined in more concrete settings such as computer graphics. Coordinates are an additional structure that, as we shall see, is treated as a property association, not an intrinsic attribute of a point. Point sets are typically infinite in the conceptual aspect, but typically finite in the computational and data aspects. Just how this "discretization" comes about and how to describe it is a central theme in our analysis framework.

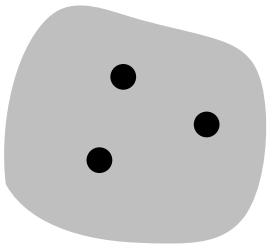


Figure 4: An infinite point set, with three points specifically identified.

Now let's consider our well as a point set. We assume our well actually exists somewhere on Earth. The points in the well are a subset of the points in the Earth. The Earth is a subset of all physical space and, following the usual assumptions of elementary physics and geology, we assume physical space is the space of three dimensional Euclidean geometry, "E3". Mathematically, E3 is a point set, and any subset of a point set is a point

set, so our well is a point set. It is an infinite point set and, at the point set level of abstraction, has no other properties, not even shape.

4.2 Topological space

The next layer of abstraction is topological space. A topological space is a point set with the notion of continuity added, but still without any notion of dimension, shape, or position. The basic idea is that if two concrete spaces that do have these features can be continuously deformed into each other, as suggested in Figure 5, then they have the same underlying abstract topological space and are said to be topologically equivalent.

Figure 5: Topologically equivalent spaces

On the other hand, if we have to discontinuously deform, that is tear or puncture, a space to transform one to the other then they are not the same, as suggested in Figure 6.

Figure 6: Spaces that are not topologically equivalent.

So a topological space is point set and, like a point set, has no dimension, shape, or position, but it does have a definite number of holes or disjoint pieces.

The mathematical machinery by which one defines continuity without having any specific definition of dimension, shape, or position is probably the most abstract of the concepts in our toolkit. We'll state the basic definitions, but rely mostly on the intuitive notion of continuous deformation whenever possible.

A topological space $\mathfrak{X}$ is a pair (X, Ω) where X is a point set and Ω is a set of subsets of X, called the open sets of $\mathfrak{X}$. The open sets satisfy the following axioms:

- any union of open sets is open,
- any finite intersection of open sets is open, and
- X and the empty set ∅ are open.

Ω is also called the topology of $\mathfrak{X}$. A map ϕ: $\mathfrak{X} \rightarrow \mathfrak{Y}$, where $\mathfrak{X}$ and $\mathfrak{Y}$ are topological spaces is continuous if the inverse image of an open set in $\mathfrak{Y}$ is always open in $\mathfrak{X}$. If ϕ is a one-to-one correspondence (bijection) and both ϕ and its inverse are continuous, then ϕ is a homeomorphism and $\mathfrak{X}$ and $\mathfrak{Y}$ are said to be homeomorphic. Homeomorphism is the

technical term for what we called a continuous deformation above. Homeomorphic and topologically equivalent are synonyms.

That's it for the basic definitions. They are deceptively simple but elaborating their meaning and implications is a task well beyond the scope of this tutorial. The reader is refered to the "Additional reading", below.

Is our well a topological space? We've already seen that the well is a subset of E3. It is well known that E3 is a topological space, perhaps the most common example of one, and a subset of a topological space is also a topological space. So our well is a topological space. Still featureless, no dimension, no shape, but we can talk about the number of holes (0) or disjoint pieces (1).

4.3 Partially ordered set

We discussed extensively the notion of "part space" in the first two tutorials in this series, Part Space and Part Spaces for Scientific Computing. The partially order set, or "poset" for short, is the fundamental mathematical concept underlying the part space metaphor.

Before we define the partially order set abstraction, we first have to recall a few concepts from set theory. The Cartesian product of two sets A and B, denoted $A \times B$, is the set of all pairs of members, one from A and one from B. The Cartesian product can be represented as a table, with one column for A, one column for B, and a row for each possible pair ($a \in A$, $b \in B$). A relation on A and B is a subset of $A \times B$. A relation can also be represented as a table, with the same columns as the Cartesian product, but only some of the rows. A relation on a single set B is a subset of $B \times B$.

With those facts in hand, we can define partially ordered set. A partially ordered set *B* consists of two components, a regular set called the base set B and a relation "≤" on B called the order relation. The set B can contain any type of object and the relation ≤ can have any type-specific definition and meaning, but it must satisfy three axioms. For all b, b', and b" members of B, ≤ must satisfy:

$b \le b$	(reflexive property)
$b \le b'$ and $b' \le b''$ implies $b \le b''$	(transitive property)
$b \le b'$ and $b' \le b$ implies $b = b'$	(antisymmetric property)

The relation is called the order relation and is traditionally represented by the symbol ≤, but the meaning of these axioms would be clearer to the beginner if it were called the inclusion relation and represented by the symbol ⊆. Subset inclusion is the prototypical example of an order relation and the axioms capture the essential behavior of inclusion.

The typical base space in the scientific computing domain is modeled as a topological space, and hence a point set. But these base spaces typically also have some sort of part structure which can be modeled as partially ordered set. So the typical base space has both a set of points and a set of parts. It's important to keep straight which set is which. Each part of the base space corresponds to a subset of the points in the base space. We

can define an order relation for the set of parts using subset inclusion in the set of points. That is, part b' ≤ part b if and only if the point subset corresponding to part b' is included in the point subset corresponding to part b.

We can use our well example to make this construction more concrete. We've already seen that the well is a point set. We can use brace notation to enumerate some of the points in this set:

well = $\{p_{b00}, p_{b01}, ..., p_{b10}, p_{b11}, ..., p_j, ..., p_{uwi}, p_{uwi+1}, p_{uwi+2}, ..., p_{df}\}$

where the p_i are each points and the meaning of the subscripts will become clear shortly. Of course the set contains a infinite number of points, so we can't fully enumerate them this way. The ellipsis ... indicates all the points we've not explicitly enumerated.

Referring to Figure 3, we see several parts identified. So in addition to the set of points, we also have a set of parts:

parts = {bore0, bore1, lower well, junction, upper well, derrick floor}

It's clear from the figure that each part of the well corresponds to a subset of the points in the well. Using the points we enumerated above:

bore 0 = $\{p_{b00}, p_{b01}, ..., p_j\}$
bore 1 = $\{p_{b10}, p_{b11}, ..., p_j\}$
lower well = $\{p_{b00}, p_{b01}, ..., p_{b10}, p_{b11}, ..., p_j\}$
junction = $\{p_j\}$
upper well = $\{p_j, ..., p_{uwi}, p_{uwi+1}, p_{uwi+2}, ..., p_{df}\}$
derrick floor = $\{p_{df}\}$

Most of the parts, bore 1, bore 2, lower well, upper well, are infinite point sets, but the junction and the derrick floor are each a subset containing a single point.

So the set of parts can be viewed as a set of subsets:

parts = $\{ \{p_{b00}, p_{b01}, ..., p_j\}, \{p_{b10}, p_{b11}, ..., p_j\}, \{p_{b00}, p_{b01}, ..., p_{b10}, p_{b11}, ..., p_j\}, \{p_j\}, \{p_j, ..., p_{uwi}, p_{uwi+1}, p_{uwi+2}, ..., p_{df}\}, \{p_{df}\} \}$

We can define an order relation for the set of parts using subset inclusion in the point set. For instance, junction ≤ bore0 because $\{p_j\} \subseteq \{p_{b00}, p_{b01}, ..., p_j\}$. Remembering from above that a relation can be described as a table, we display the entire order relation in Table 1.

Table 1: Order relation on well parts.

Lesser Part	Greater Part
derrick floor	derrick floor
derrick floor	upper well
derrick floor	well
junction	junction
junction	bore 0
junction	bore 1
junction	lower well
junction	upper well
junction	well
bore 0	bore 0
bore 0	lower well
bore 0	well
bore1	bore1
bore1	lower well
bore 1	well
upper well	upper well
upper well	well
well	well

Table 2: Covering relation on well parts.

Covered Part	Covering Part
derrick floor	upper well
junction	bore 0
junction	bore 1
junction	upper well
bore 0	lower well
bore1	lower well
lower well	well
upper well	well

The order relation is a complete but not very efficient representation of the essential information. We can use the axioms of the order relation to reduce the size of the table, at least for finite posets. Since the reflexive axiom requires $b \leq b$ for all $b \in B$, there's no need to explicitly store (derrick floor, derrick floor), (junction, junction) and other such rows in the table. Similarly, the transitive axiom means that since (junction, bore 0) and (bore 0, lower well) are in the table, there's no need to explicitly store (junction, lower well) in the table either.

Removing all such transitive and reflexive members of a relation produces a different relation, called the transitive, reflexive reduction of the original relation. In the case of an order relation, the transitive, reflexive reduction is called the covering relation. We will denote the covering relation with the symbol "$\prec\!\!\!-$". If $b' \prec\!\!\!- b$, we say that b' is covered by b, or b covers b'.

We can also define the covering relation directly. First we define the strict order relation "<" by $b' < b$ if and only if $b' \leq b$ and $b' \neq b$. Then we can define $b' \prec\!\!\!- b$ if and only if $b' < b$ and there is no intervening b" such that $b' < b'' < b$. The "no intervening" requirement in the definition implies that for some infinite posets the cover relation may be empty. For instance, for real numbers x and y with $x < y$, there is always an intervening number z such that $x < z < y$ so the covering relation is empty. But for finite posets the covering relation always exists and is both determined by and determines the order relation.

The covering relation for our well parts poset is shown in Table 2. As even this very small example shows, the cover relation is much smaller than the order relation.

We can use either the order relation or covering relation to represent a finite poset as a graph. We represent each member of the poset as a node in the graph and we create a link from b to b' if and only if $b \leq b'$ or b covers b', respectively. The result is a directed acyclic graph, a well-known, powerful, and efficient data structure which we can use to represent a partially ordered set on the computer.

We can also use either of these graphs to visualize a poset. Each node is represented by a box. For the order relation graph, the box is positioned so that if there is a link from b to b', b' is higher on the page than b. The cover relation links point in the opposite direction, if b covers b', then $b' \leq b$ so for the covering relation b' is positioned lower on the page. The order and covering relation graphs for the well parts poset are shown in Figure 7 and Figure 8, respectively. As one would expect from the size of the table representation, the covering relation graph is considerably simpler. The covering relation graph is called a Hasse diagram in mathematics and we will use only it, not the order relation diagram, to visualize posets.

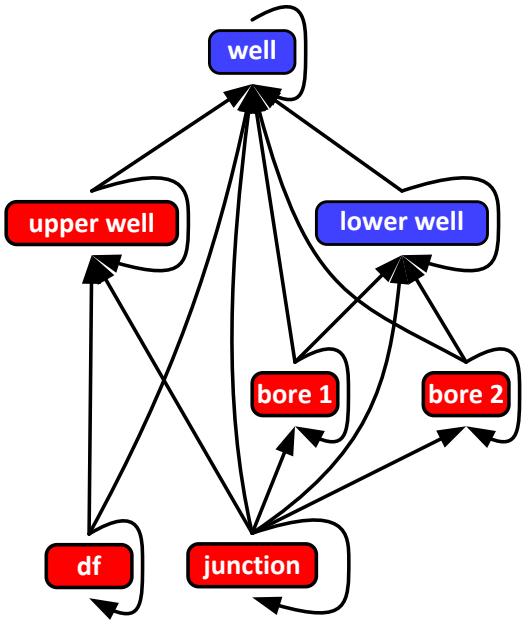


Figure 7: Graph of order relation for well parts poset

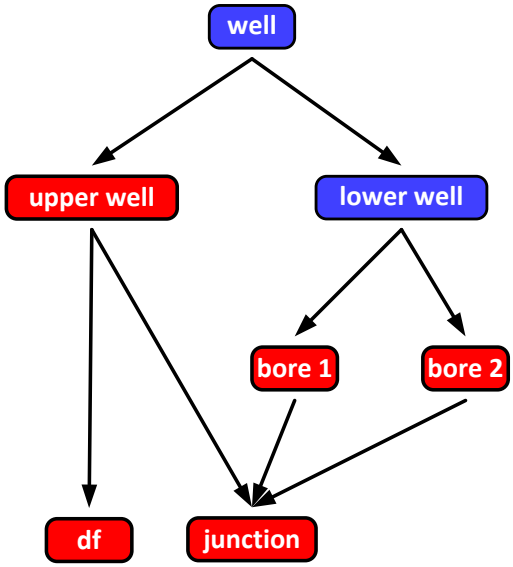


Figure 8: Graph of cover relation (Hasse diagram) for well parts poset.

Finally, it useful to observe that our well has two kinds of parts. A composite part is a part that is equivalent to a collection of other parts while a basic part is not. Each of the well parts is a subset of the points in the well, but the subset corresponding to the lower well is precisely the union of subsets corresponding to bore 0 and bore 1. Similarly, the well itself is equivalent to the union of the lower well and the upper well. On the other

hand, bore 0, bore 1, the junction, the upper well, and the derrick floor are basic parts. None of the subsets associated with these parts can be expressed as the union of other parts. The basic parts, which are colored red in Figure 8, form a set of building blocks from which we can construct composite parts.

4.4 Summary

The base space role in the conceptual aspect is played by some sort of set, typically a point set. The base space typically has two additional layers of structure. It is typically a topological space and has a partially ordered set of parts.

4.5 Additional reading

General set theory is the foundation for all base space types (and fiber space and section space types as well).

(Halmos, 1974) provides an accessible introduction to the basics of set theory.

(Rosen, 1995) covers both sets and posets, as well as many other topics, at an introductory level specifically intended for applications in computer science.

Point sets and topological spaces are typically treated together.

(Janich, 1984) provides a relaxed introduction to topology, emphasizing intuitive, graphical motiviation of each concept before going into more formal definition.

(Munkres, 1975) is an intermediate level introduction. More formal than Janich but still introductory and covers more material.

(Fuks, et al., 1984) is a much more thorough and formal treatise. It is for the beginner only in the sense that it supposes no knowledge of topology on the part of the reader and covers everything from the fundamental concepts to advanced topics. But its choice of topics is well matched to the requirements of the sheaf data model and when you really need to know the details, this is the place to look.

Partially ordered sets and lattices have historially been of interest mostly in pure mathematics but have become more important to applications in the last two decades.

(Davey, et al., 2002) is now the standard introductory text in the field. It is readable, if perhaps with some effort, and specifically addresses applications, especially in computer and information science.

(Birkhoff, 1995) is the classic reference on partially ordered sets and lattices by a founder of the field. It covers a wide range of topics but is intended for academic mathematicians.

5 Conceptual aspect for the fiber space role

The fiber space role in a property association is a space of all possible values of some property. In principle, any type can be used as a fiber space, including the point set, topological space, and poset types that we've just seen are important in the base space role. In scientific computing applications however, the fiber space role is typically played by one of the family of property types defined by mathematical physics: scalar types, vector types, and tensor types.

These types are all specializations of the abstract vector space type and can be organized into an inheritance hierarchy, the core of which is shown in Figure 9. The arrows here represent inheritance, not the order or the covering relation, but we will see in section 11.1 that these concepts are related.

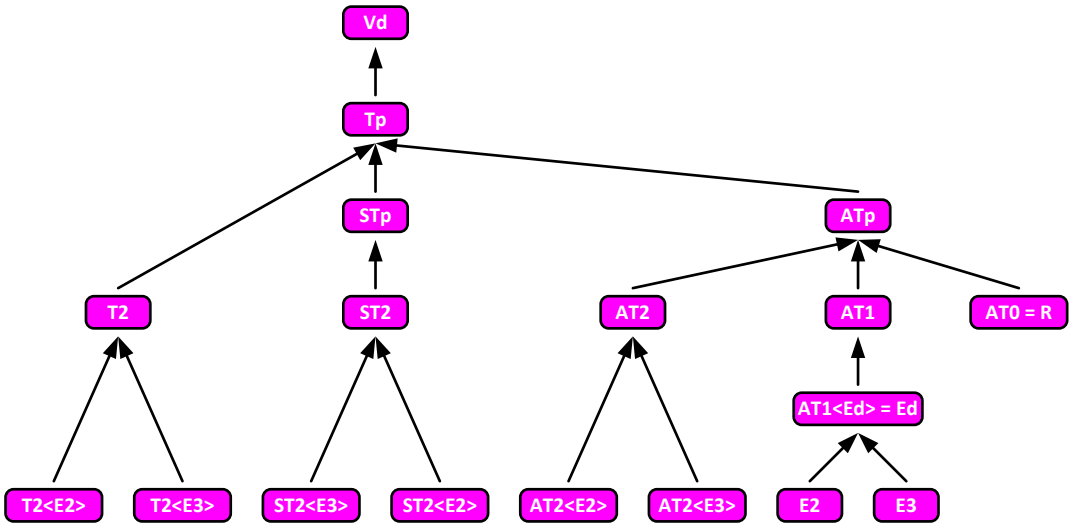


Figure 9: The core of the physical property inheritance hierarchy.

The central types of this hierarchy are:

- Vd: abstract vector of dimension d
- Tp: general tensor of degree p
- ATp: antisymmetric tensor of degree p
- Stp: symmetric tensor of degree p

As shown in the figure, the hierarchy differentiates on symmetry and degree. In addition, since a tensor is fundamentally a map on vectors and a tensor space is defined with respect to the specific kind of vector it maps, the hierarchy differentiates on vector type as well. The figure shows only a portion of the complete hierarchy supported by the Sheaf System, the portion corresponding to degrees 0 through 2 and vector spaces E2 and E3.

Although the mathematical structure of this hierarchy is critical to the design and implementation of the Sheaf System class libraries, applications typically only use the leaf classes in the core of the hierarchy, as shown in the figure.

5.1 Additional reading

The abstract vector space aka linear algebra is the base type for all the property types of mathematical physics.

(Janich, 1994) is a friendly introduction to the subject.

The extensive family of physical property types is based on the theory of multilinear algebra, which is built on top of linear algebra. Unfortunately there does not seem to be a reference that is both accessible and comprehensive. So we have to look in several places.

(Bishop, et al., 1980) Chapter 2, "Tensor Algebra" is the closest thing to an accessible and complete treatment. If you can only afford one reference, this is the one. But it's a Dover book, so you will be able to afford others! It also includes chapters on set theory, topology, and other relevant subjects.

(Crampin, et al., 1986) Chapter 4, "Volumes and Subspaces" covers exterior algebra including multivectors and multiforms. All the other chapters address relevant topics as well, so buy this one with the money you saved because Bishop and Goldberg was a Dover book.

(Koenderink, 1990), section 3.4 "The myopic view" gives an intuitive account of linear and multilinear algebra with lots of pictures, but few formal definitions. It's very helpful when combined with the formal treatment in (Crampin, et al., 1986).

(Abraham, et al., 1983), Chapter 5 "Tensors" and Chapter 6 "Differential Forms". This book can hurt your head, but as with Fuks and Rokhlin for topology, if you really need to know all the details, this is often the place to go.

6 Conceptual aspect for section role

We introduced the notion of section as an association between an entity that carries some property, which we referred to as the base space, and the collection of all possible property values, which we called the fiber space. A section associates a member of the fiber space with each member of the base space.

There are two mathematical formalisms to describe such associations. The more familiar formalism is the map formalism. The other is the fiber bundle formalism. Each formalism defines its own terminology for the association, as shown in Table 3.

Table 3: Terminology for property associations

Formalism	Entity	Property	Association
map	domain	range	map or function
fiber bundle	base space	fiber space	section

Our terminonology is derived from the fiber bundle formalism. The fiber bundle formalism is more general than the map formalism and, as we will show, can deal with problems the simple map formalism can not. But in some respects, the fiber bundle formalism is built on top of the map formalism, so we have to first describe some basic features of maps.

6.1 Map formalism

The domain and range of a map are each assumed to be a set. The map associates a member of the range with each member of the domain and we usually think of a map using an active metaphor. The map "sends" each member of the domain to some member of the range. This action is typically denoted with an arrow. If ϕ is a map, B is its domain and F is its range, we write:

$\phi: B \rightarrow F$

There is another, more static description for a map. A map ϕ must specify a member of F for each member of B. So if B has n members, we need an n-tuple of members of F:

$\phi = (f_{b1}, f_{b2}, ..., f_{bn})$ where $B = \{b_1, b_2, ..., b_n\}$

We've already introduced the binary Cartesian product in section 4.3. In particular, we introduced the Cartesian product of a set with itself. The Cartesian product of F with itself, $F \times F$, is the set of all pairs of members of F:

$F \times F = \{(f_i, f_j)\}$

We can iterate the Cartesian product to any number of factors, $F \times F \times F$, $F \times F \times F \times F$, etc. If we iterate the product of F with one factor for each member of another set, for instance the domain B, we get a construction called the Cartesian power, denoted F^B. F^B is the set of all n-tuples of members of F, with one component in the tuple for each member of B:

$F^B = \{(f_{b1}, f_{b2}, ..., f_{bn})\}$

Comparing this with the description of our map ϕ as a n-tuple, we can see that each n-tuple is a map and F^B is the set of all possible maps from B to F. We previously introduced the term "section space" for the set of all possible associations between a particular domain and range. So our section space is in fact the Cartesian power, fiber space to the base space power, at least for associations that can be described as maps.

Table 4: The Cartesian power $bool^{Z3}$

$bool^{Z3}$		
0	1	2
bool	bool	bool
F	F	F
F	F	T
F	T	F
F	T	T
T	F	F
T	F	T
T	T	F
T	T	T

Table 5: The Cartesian product Z3 × bool

Z3 x bool	
Z3	bool
0	F
0	T
1	F
1	T
2	F
2	T

Some examples will help clarify this notion. First let's take a simple, finite domain, a set of 3 integers:

Z3 = {0, 1, 2}

and a simple range, the set of truth values:

bool = {F, T}

The section space of all possible maps f: Z3 → bool is the Cartesian power:

$bool^{Z3}$ = bool × bool × bool

We can represent it as a table, as shown in Table 4, with the columns labelled by the members of B. Each row in this table is a map from Z3 to bool.

Beginners often confuse the Cartesian power with the Cartesian product Z3 × bool. As shown in Table 5, each member of the Cartesian product is a pair (i ∈ Z3, v ∈ bool).

Now let's examine a more complex case: temperature on the surface of the Earth. The domain is the surface of the Earth, which we will take to be a sphere, S2, an infinite set of points. The range is the set of all temperatures, which we will take to be the set of real numbers, R. The section space is the Cartesian power R^{S2}. We can still, at least informally, think of this as a table. But since there are an infinite number of members in the base space, the power has an infinite number of factors and the table has an infinite number of columns. An infinite number of columns means an infinite number of rows.

Even worse, in the finite case we could enumerate the members of the domain and label the columns with the members. How do we even identify the members of the domain in the infinite case? We've tried to suggest all this graphically in Table 6.

Table 6: Cartesian power R^{S2}

R^{S2}		
?	?	...
R	R	...
−∞	−∞	...
...	...	...
0.0	0.0	...
...	...	...
∞	∞	...

The solution to our problem is to use another property association to label the points in the domain. To do this, we need a special kind of property association. The general property association associates some property value with each member of the domain, but it may associate the same property value with more than one member of the domain. What we want to do is use a property value to uniquely identify each point in the domain, so we need an association that is a one-to-one correspondence. We'll call such a property "coordinates" and refer to a point by the value of the coordinates at the point.

So with that goal in mind, let's try to establish some coordinates for the surface of a sphere. Figure 10 shows ordinary latitude-longitude coordinates for a sphere. But lat-lon has well known problems as a coordinate system. There is not a one-to-one correspondence between the line (lat = 90, lon = *) and the single point at the pole on the sphere. Similarly, every point on the international date line gets two lat-lon pairs. Furthermore, latitude and longitude define a local basis for directions at each point, with one basis vector pointing towards increasing latitude and one towards increasing longitude, as indicated by the red basis vectors in the figure. The local basis vectors are also not one-to-one and continuous at the poles.

The problems with latitude and longitude as coordinates are symptomatic of a fundamental mathematical problem: it is not always possible to cover an entire domain with a single one-to-one, continuous coordinate system. Because of this fundamental fact, we can not always represent property associations as simple maps. The fiber bundle formalism was developed specifically to overcome this problem.

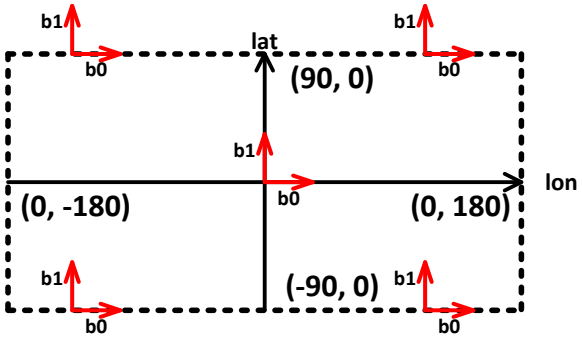
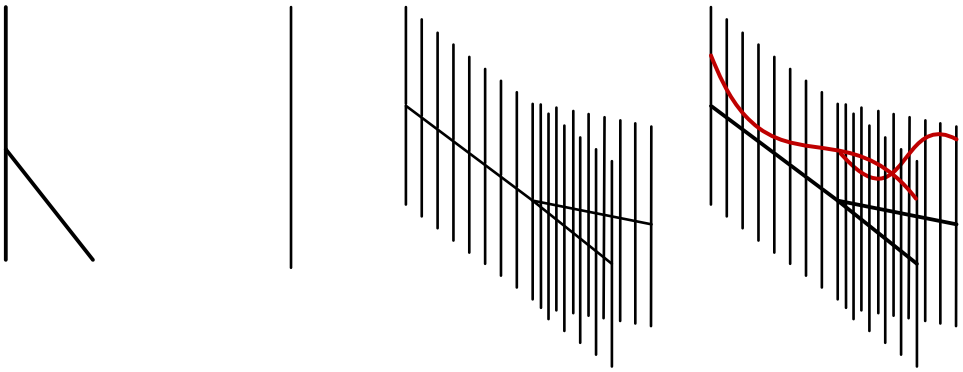


Figure 10: Latitude and longitude coordinates for S2

6.2 Fiber bundle formalism

The fundamental idea of fiber bundle theory is to decompose the domain into patches and reformulate the notion of map as a collection of simple maps, one on each patch. The first step in this program is to recast the notion of a simple map in more geometric terms.

We'll return to our branched well to show how this reformulation proceeds. We'll use the well as a domain and assume we have a scalar property such as temperature.



base space = well **fiber = R^1** **bundle = well x R^1** **section $\subset$ well x R**

Figure 11: The notion of a "trivial" fiber bundle.

Figure 11 shows the fiber bundle interpretation of this example. The domain, referred to as the base space in fiber bundle terminology, is the well. The range, or fiber space, is the real number line. The fiber bundle is the Cartesian product of the base space and the fiber space. We can visualize this Cartesian product as consisting of a copy of the fiber space attached to each point in the base space. Since there are an infinite number of points in the base space, we have an infinite number of fibers, a "bundle" of them, but the figure only shows a few. Now the value of the map corresponds to a particular point on each

fiber and collectively these points define a cross-section, or just section for short, of the bundle. So a map is referred to as a section of a fiber bundle.

So far, this is just a reformulation of map. Now we decompose the base space into a collection of overlapping patches, as shown in Figure 12. The patches are explicitly chosen so that each can be equipped with coordinates and the property association is a simple map on each patch.

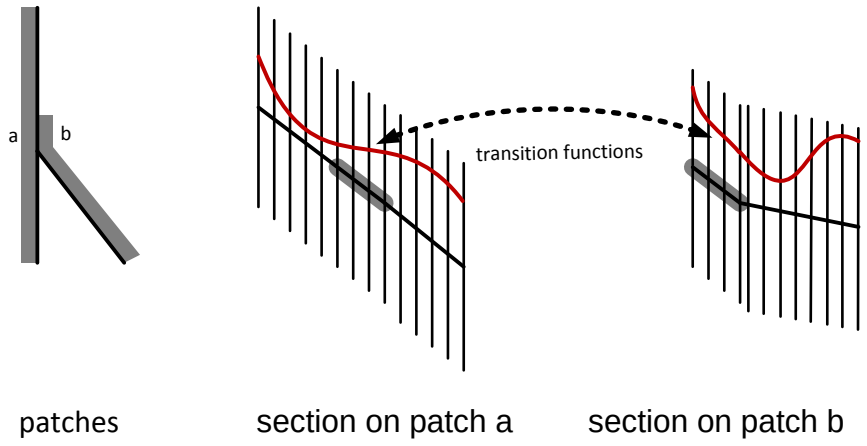


Figure 12: The notion of a non-trivial bundle.

Usually we want our property association, the section, to be single valued. That is, there should be a single value for the property at each point in our base space. But wherever the patches overlap, we will have a property value in each patch. The fiber bundle formalism explicitly assumes that each patch may use a different scale, calibration, or reference orientation for representing the property. Sometimes the mathematics requires the use of such multiple reference frames for the property. But even when such frames are not required theoretically, they may be desireable for practical reasons. In any case, the fiber bundle is equipped with "transition functions" for converting the property values in one patch to the reference frame of any overlapping patch. This restores the notion of the section being single valued, even when it is multiply stored on overlapping patches.

6.3 Summary

In simple cases, a property association is represented mathematically as a map. The section space for a simple map is the Cartesian power of its fiber space to the base space power. The fiber bundle formalism is designed to handle complex cases where the base space must be decomposed into patches and a property association represented as a collection of simple maps, one on each patch.

6.4 Additional reading

The map or function concept is treated in most discussions of set theory.

(Halmos, 1974) Section 8 "Functions" succinctly covers the essentials. He defines the Cartesian power Y^X as just notation for the set of all functions from X to Y, which is perhaps a bit too succinct.

(Munkres, 1975) Chapter 1, section 2 "Functions" also covers the essentials. Section 5 "Arbitrary Cartesian Products" takes some care to define products indexed by both finite and infinite sets. The Cartesian power pops out of the discussion unannounced right at the end of the section.

Fiber bundle theory became part of mainstream mathematical physics in the 1970's and several introductory texts followed a decade or so later. None of the available references are simple. An introductory paragraph in (Crampin, et al., 1986) nicely sums up the fundamental difficulty with fiber bundles: "It will be clear that the definition of a fibre bundle (as distinct from the object itself) is a fairly complex matter."

(Nash, et al., 1983) Chapter 7 "Fibre Bundles and Further Differential Geometry" presents the basics, with pictures. Note the British (as opposed to American) spelling of fiber.

(Crampin, et al., 1986) Chapter 14 "Fibre Bundles" provides a more precise treatment, but no pictures.

(Abraham, et al., 1983) chaper 3, section 3 "Vector Bundles" gives a precise treatment of vector bundles, the most important type of fiber bundle, with numerous pictures.

7 Computational aspect for base space role

The computational aspect focuses on reformulation of the entities of the conceptual aspect for representation on the computer. For base space entities, the traditional notion of computer representation is that we somehow "discretize" or "sample" the base space. A more accurate description is that we refine the poset of parts.

Refinement means to subdivide each part into smaller parts, the union of which equals the original part. We don't in principle discard any of the points in the original part, in contrast to what the terms discretization or sampling suggest. Each basic part in the original poset becomes a composite part equivalent to the union of the new, finer-grained basic parts. The collection of new basic parts is usually called a mesh and the refinement process is called meshing or mesh generation.

Refinement introduces more basic parts into the poset of parts. The size and number of basic parts in a mesh is driven by the need for numerical accuracy; higher accuracy requires more, smaller basic parts. In realistic cases, meshing introduces many more basic parts, typically 10^3 to 10^9 or more times as many. Meshing may also introduce new composite parts. For instance, the entire mesh may be decomposed into "domains" for parallel processing.

We can illustrate the meshing as refinement notion using our well example. We'll have to use just a few basic parts in order to make drawing the pictures feasible. Figure 13 shows

a geometric view of meshing the well base space while Figure 14 shows the refinement as a covering relation graph (Hasse diagram). As you can see, each old basic part is refined into multiple new basic parts, except the derrick floor and junction, which only contained a single point each to start with. Each old basic part becomes a new composite part.

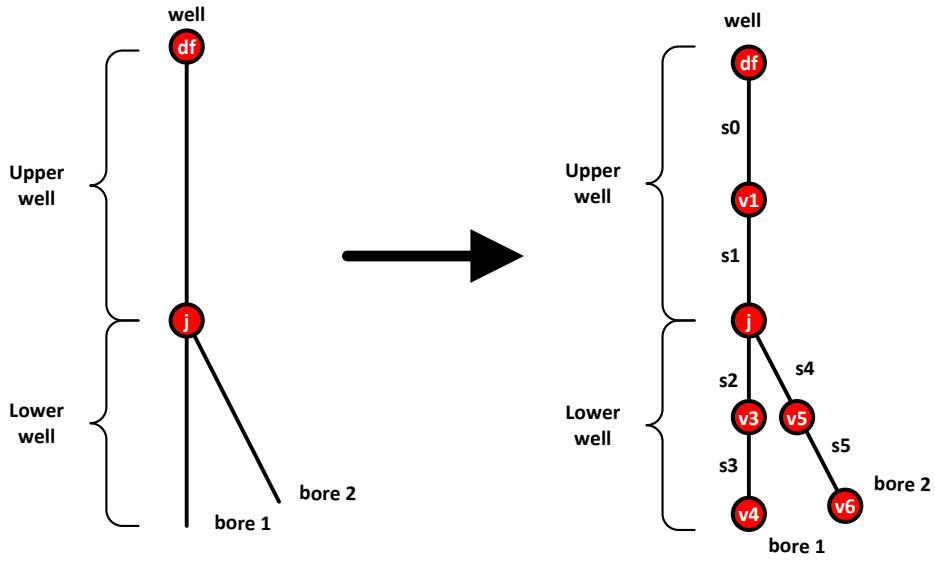


Figure 13: Geometric view of meshing as part refinement.

Refinement increases the number of possible composite parts exponentially. Here we've only increased the number of basic parts by a small factor, but as we said above, meshing increases the number of basic parts by a very large factor in practice. The number of possible composite parts becomes enormous, so we need a mathematical tool to manage all the potential parts. As we've already seen in the previous tutorials, that tool is part space.

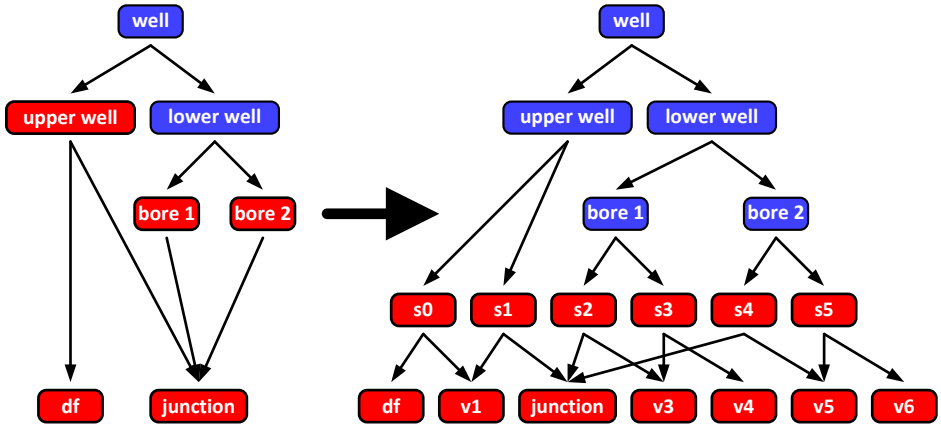


Figure 14: Covering relation graph view of meshing as part refinement.

7.1 Finite distributive lattice

When we introduced the partially ordered set concept in section 4.3, we said it was the fundamental mathematical concept underlying the part space metaphor. True enough, but it is not the only mathematical concept behind the metaphor. Associated with every finite poset is another poset, a special kind of poset called a finite distributive lattice.

Remember that our part space metaphor identified basic and composite parts and organized them into a partially ordered set. We defined part space as the set of all distinct assemblies that can be generated from the set of basic parts. The precise mathematical concept corresponding to part space is the finite distributive lattice associated with the poset of basic parts.

Given any poset P, any member $p \in P$, and any subset $S \subseteq P$, then if $s \leq p$ for all $s \in S$, we say that p is an upper bound for S. If the set of all upper bounds for S has a unique least member, we call it the least upper bound or join of S. The join of two members p and p' will be denoted $p \vee p'$ and the join of a subset S will be denoted $\vee S$.

Similarly, if $p \leq s$ for all s, we call p a lower bound for S and if the set of all lower bounds has a unique greatest member we call it the greatest lower bound or meet of S. The meet of p and p' will be noted $p \wedge p'$ and the meet of S will be denoted $\wedge S$.

A lattice is a poset in which the join and meet exist for any finite subset S. A complete lattice is a lattice in which join and meet exist for every subset S, including infinite subsets if the lattice has them. Every finite lattice is thus a complete lattice.

Every complete lattice has a unique greatest member, called top, which is the join of the entire lattice. Similarly it has a unique least member, called bottom, which is the meet of the entire lattice.

A distributive lattice P satisfies the distributive law:

$p \wedge (p' \vee p'') = (p \wedge p') \vee (p \wedge p'')$

for all p, p', and p'' in P.

An example will help clarify these definitions. As always, we'll have to keep the example small to make the diagrams manageable. The well poset is already too large to draw the entire part space associated with it, so we'll have to use an even smaller example, the two segment polyline that we first saw in the Part Spaces For Scientific Computing tutorial. Figure 15 shows the polyline and its poset of basic parts, while Figure 16 shows the entire finite distributive lattice for the polyline.

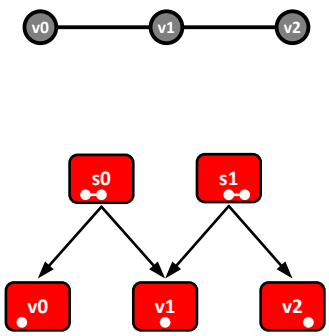


Figure 15: Poset of basic parts for polyline.

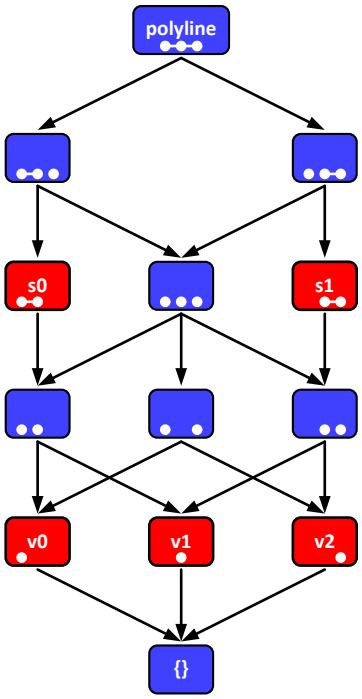


Figure 16: Finite distributive lattice for polyline.

We've described part space as the set of all distinct assemblies that can be generated from a set of basic parts and we've seen previously that the set of all distinct assemblies is not in general the set of all subsets. So what subsets correspond to distinct assemblies? The answer is down sets. A down set is a subset of a poset that is closed under going down in the graph. When ever a down set contains a member p, it also contains all members less than p. The down set (in the poset of basic parts) of a basic part is the part and all the parts below it in the graph, all the parts one can reach by following links downward. The union of two down sets is a down set and so is their intersection.

The fundamental theorem of the theory of finite distributive lattices, the Birkhoff representation theorem, says that every finite distributive lattice is equivalent (isomorphic) to the set of down sets of its poset of basic parts. The set of down sets of a poset P is traditionally denoted $\mathcal{O}(P)$. A basic part in a finite distributive lattice $\mathcal{L}$ is formally referred to as a join irreducible member and the set of join irreducible members is denoted $J(\mathcal{L})$. The formal statement of the Birkhoff representation theorem is thus that for any finite poset P there exists a finite distributive lattice $\mathcal{L}$ such that $\mathcal{L}$ is isomorphic to $\mathcal{O}(P)$ and P is isomorphic to $J(\mathcal{L})$.

Figure 17 depicts the Birkhoff representation theorem for the polyline of Figure 15. The memberwise interpretation of this equivalence is that every member (node in the left hand side of Figure 17) of a finite distributive lattice can be interpreted as the collection of basic parts in its down set (node in right hand side of Figure 17). In practical terms, this means we can think of any part in two ways: either as a collection of subparts (right

hand side), or as the single part we get when we join the subparts together (left hand side). If the part is already a basic part, the join just gives us the basic part again.

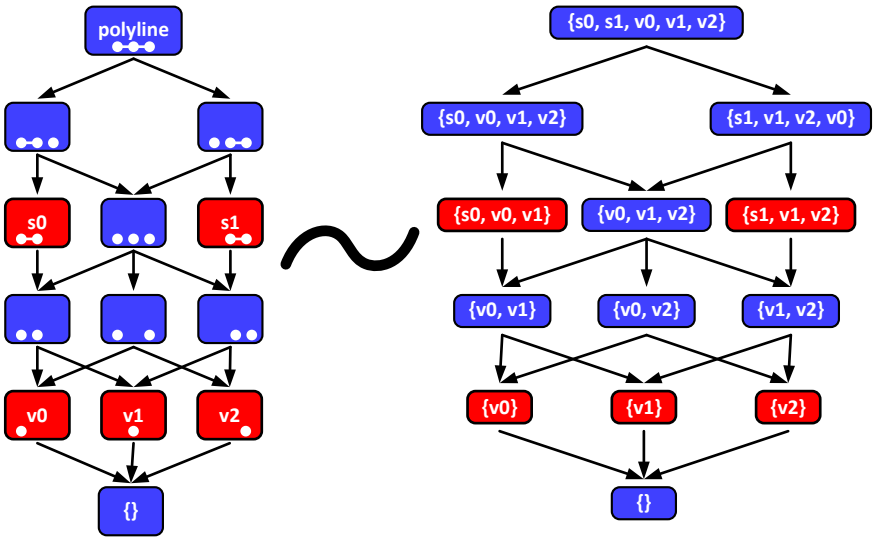


Figure 17: The Birkhoff representation theorem.

As we said above, the technical term for basic part is "join-irreducble member". We'll refer to this by the acronym "JIM" and pronounce it like a person's name. A basic part is a member of the lattice that is not equivalent ("reducible") to the join (assembly) of other parts. It is a part that is greater than the "sum" of its subparts. We can see this in either graph in Figure 17. Each basic part (JIM) has a single member immediately below it in the graph and this single member corresponds to the join of the subparts of the JIM. In practice, the fact that a JIM is greater than the sum of its parts is the key to interpreting an application structure as a finite distributive lattice. A JIM represents something not entirely defined by its part decomposition. It introduces something new into the structure. For instance, the v0, v1, and v2 parts each introduce a single point while s0 and s1 each introduce the infinite number of points in their respective interiors.

There isn't a specific mathematical term for composite part. But a composite part is not join irreducible, it can be represented as the join of some basic parts, so it must be a "join reducible member". We'll convert that to the acronym JRM and pronounce it like "germ". A JRM always has multiple members immediately below it in the graph and it is the smallest member that contains these subparts. It is precisely the sum of its subparts. In practice, a JRM is something entirely defined by its part decomposition, a grouping construct.

We've introduced the essential features of finite distributive lattices and we're ready to move on. But before doing so, we'll emphasize the importance of the concept. Spatial decompositions are ubiquitous in scientific computing. Every such decomposition defines a set of basic parts that can be represented as a finite poset. The finite distributive lattice gives us a complete space for describing these decompositions and the composite parts that can be constructed from them.

7.2 Cell complex

Most of the spatial decompositions encountered in practice, in other words most meshes, are in a category of decompositions called cell complexes.

The definition of a cell complex follows a common pattern in topology: define or analyze a topologically complex object by decomposing it into pieces that are topologically simple, along with maps of some kind that determine how the simple pieces are "glued" together to form the whole complex structure. This pattern occurs repeatedly in various similar, but not identical, forms. We've already seen one instance of this pattern in the definition of fiber bundle: a map on a topologically complex base space is decomposed into a collection of simple maps on simple patches and glued back together with transition functions. The definition of cell complex is another variation of the pattern.

To describe what a cell complex is, we need to introduce some machinery. A partition P of a topological space $\mathfrak{X}$ is a decomposition of $\mathfrak{X}$ into a set of disjoint subspaces (subsets) that completely cover $\mathfrak{X}$, that is, the union of the subspaces is the whole space. Every point of the space is thus contained in exactly one member of the partition.

The unit ball D^n in the n-dimensional Euclidean space E^n, is the set of points with distance from the origin ≤ 1 and the unit sphere S^{n-1} is the set of points with distance from the origin = 1.

With these preliminary definitions in hand, we can define a cell complex. A cell complex or cell decomposition of a topological space $\mathfrak{X}$ consists of 3 components:

- a partition C of $\mathfrak{X}$ into a finite number of subspaces called cells;
- a map d: C $\rightarrow$ Integer which assigns each cell a dimension; and
- a collection of maps, one for each cell, called the characteristic maps.

The characteristic map ϕ_c for each cell c must have the following properties:

1) ϕ_c is a continuous map from the unit ball $D^{d(c)}$ to $\mathfrak{X}$;
2) ϕ_c maps the interior of the unit ball homeomorphically to the cell c, (remember from section 4.3 that a homeomorphic map means domain and range can be continuously deformed into each other); and
3) ϕ_c maps S^{n-1}, the boundary of the ball, to the union of a collection of cells with lower dimension.

We've limited our cell complex to a finite number of cells. Most mathematical texts define a more general type of complex, a CW-complex, which allows an infinite number of cells if certain additional axioms are satisfied. We will be concerned only with finite cell complexes, so we can ignore the additional complexity.

All this is much harder to state in words than it is to visualize. Figure 18 shows a simple 1D cell complex consisting of an open line segment (it doesn't contain its end points) and two vertices. D^1, the unit ball in 1D, is just the closed line segment [-1.0, 1.0] in E^1 and S^0, the boundary of the ball, is just the two end points in E^1. The characteristic map for the segment takes the interior of the ball to the segment and S^0 to the two vertices. D^0 consists of a single point and S^{-1} is empty, so the characteristic map for each vertex just maps the only point in D^0 to the vertex.

Similarly, Figure 19 shows a 2D cell complex consisting of an open quad, 4 open segments, and 4 vertices. D^2 is just the unit disk while S^1 is the unit circle. The characteristic map for the quad takes the interior of the disk to the open quad and the unit circle to the union of the 4 segments and 4 vertices. The characteristic maps for each edge and each vertex are the same as in the 1D example.

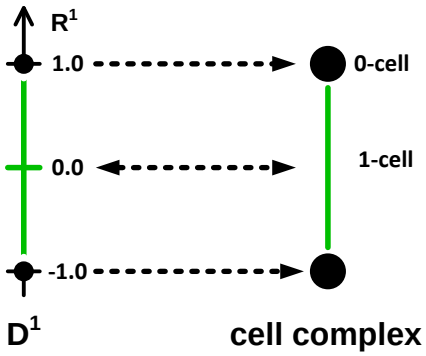


Figure 18: 1D cell complex

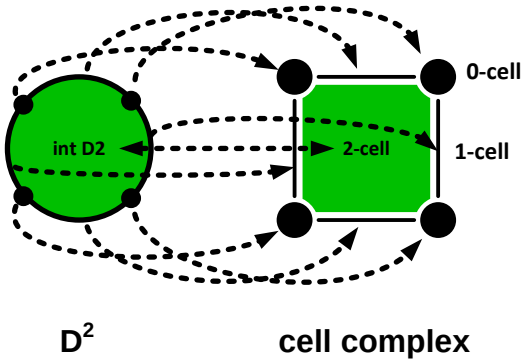


Figure 19: 2D cell complex

We said above that the pattern is to first decompose a complex object into pieces that were topologically simple, then provide maps that glue the pieces together. The two figures above show how axiom (2) of the characteristic maps guarantees that each piece is topologically simple: each cell must be chosen such that it can be mapped homeomorphically to the interior of a ball, which is the definition of "topologically simple".

But what about the second part of the pattern, the maps that glue the pieces together? Well, the characteristic maps are the only maps in the definition of a cell complex, so somehow they must be the glue. Indeed they are and it is axiom (3) that makes them so. The cells and characteristic maps can be chosen so that a portion of the boundary in each of two or more d-cells gets mapped to the same lower dimensional cells, thus gluing the two d-cells together, as shown in Figure 20.

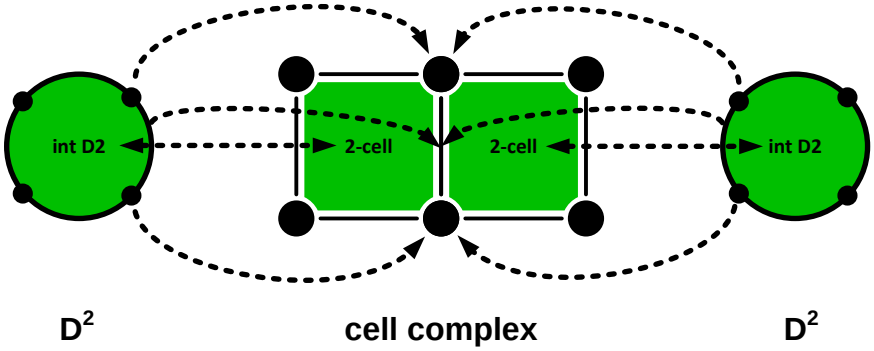


Figure 20: Characteristic maps glue cells together.

The cell complex concept is a useful tool in the computational aspect of our domain analysis framework for two reasons. First, as we've already seen in both the Part Spaces For Scientific Computing tutorial and in section 6.2 of this tutorial, we need coordinates on our base space if we are going to define sections. The characteristic maps of a cell complex provide local coordinates. They allow us to refer to any point in a cell by specifying a vector in E^n. Second, the cell complex formalism provides a rigorous bridge from the computational notion of mesh to the formal, abstract topology of the base space. We won't go into the details, but the cell complex determines the topology.

We can cast the cell complex concept into an even more useful form however. Note that since a cell complex is a partition, the cells of a cell complex, except for the 0-cells, are all open sets, they don't include their boundary. For each cell we can form a closed cell by taking the union of the cell and its boundary. The boundary of the cell is $\phi_c(S^{d(c)-1})$ and characteristic map axiom (3) requires it to be a union of cells of lower dimension. In other words, although the cells of the complex are all disjoint, the closed cells overlap at their boundaries. Each closed cell includes lower dimensional closed cells in its boundary.

The closed cells of a cell complex can be treated as a partially ordered set, with point set inclusion as the order relation. Each closed n-cell covers the closed n-1-cells in its boundary. Axiom (3) of the characteristic map definition is explictly represented by the down set of a cell, as shown in Figure 21. The gluing implied by axiom (3) is represented by two or more n-cells both including the same n-1-cell, shown in green in the figure. More formally, such "shared inclusion" between cells corresponds precisely to the meet of the cells.

Finally, the inclusion of each n-1-cell in the boundary of an n-cell can be expressed as a map from the local coordinates of the n-1-cell to the local coordinates of n-cell and these maps can be viewed a sitting on the links of the covering relation graph. The inclusion maps determine the characteristic maps of the cell complex.

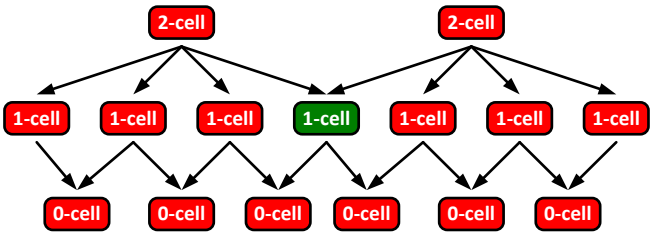


Figure 21: The poset of closed cells corresponding to the cell complex of Figure 20.

The poset of closed cells with inclusion maps is an effective computational representation of the cell complex and hence rigorously encodes the abstract topology of the base space. The finite distributive lattice associated with the poset provides a complete space for representing and managing composite parts assembled from the cells.

7.3 Additional reading

Finite distributive lattices.

(Davey, et al., 2002) Chapter 2 introduces general lattices. Chapter 5 treats finite distributive lattices, including the Birkhoff representation theorem.

(Birkhoff, 1995) Chapter III section 3 gives you the Birkhoff representation theorem from Birkhoff himself. But as mentioned above, it is intended for academic mathematicians.

Cell complexes.

(Janich, 1984) Chapter VII gives a very readable introduction to CW-complexes with many helpful pictures.

(Fuks, et al., 1984) Chapter 2 addresses "Cellular Spaces", which is what Fuks and Rokhlin call CW-complexes. As with most topics in their book, the treatment is more thorough and detailed than Janich and correspondingly less readable. But sometimes you need all the details.

8 Computational aspect for fiber space role

We saw in our discussion of the conceptual aspect that any type could be used in the fiber space role, but the vector space types from mathematical physics were particularly important. There is relatively little reformulation required to represent any of these types on the computer.

We can represent any general entity type as a class using object-oriented programming techniques. An entity is then represented as an instance of the class. All the machinery of object-oriented programming applies. For instance, attributes of the entity can be represented as data members, the operations as member functions, and inheritance can be used to represent is_a relationships between types.

These techniques apply in particular to the vector space hierarchy of physical property types introduced in Figure 9. Each type can be represented as a class; the resulting class hierarchy mirrors the type hierarchy of Figure 9. The interior nodes in the tree - Vd, Tp, ATp, STp, etc - are represented by abstract classes while the leaves - E3, T2_E3, AT2_E3, ST2_E3, etc - are represented by concrete classes.

A convenient fact when implementing these classes is that every type is_a abstract vector (Vd) and any vector can be represented by its components relative to a basis. The components can be represented directly as data members.

8.1 Additional reading

The references listed in section 5.1 for the conceptual aspect apply to the computational aspect as well.

9 Computational aspect for the section role

We saw in the conceptual aspect of the section role that in simple cases we could treat a property association as a map. We also saw that this wasn't always possible. In more complex cases we needed the fiber bundle formalism which treated the association as a collection of simple maps. We now have to consider how to reformulate the notion of map so we can represent it on the computer.

Representation of a simple map on a finite base space is straightforward. Assume we have a map from a base space B to a fiber space F, μ: B $\rightarrow$ F. We saw in the conceptual aspect that the section space for B and F, the space of all maps from B to F, is the Cartesian power F^B, which can be represented as a table with a column for each member of B. As long as the base space is finite, as in Table 4, we can represent this table directly. We create a row for each map we are interested in. We can just store this row in memory and evaluate the map at a given member of B by just retrieving the appropriate value from the row.

Representation of even a simple map is not so striaght-forward if the base space is an infinite set. As we saw in the conceptual aspect, the corresponding table has an infinite number of columns, so we can't store even a single row. How do we proceed?

The common practice approach is to divide and conquer. We subdivide the base space B into smaller pieces. We then approximate the map on each piece using some finite collection of stored data and an evaluation method that computes the value of the map when needed, using the stored data. The sheaf data model formalizes and generalizes this approach.

9.1 Coordinates

The first step in this formalization is to provide a general mechanism for referring to the points in the base space. We saw previously that this requires a coordinate system and that we can not in general cover the entire base space with a single coordinate system. We saw in the conceptual aspect for the base space role that the base space is a

topological space with a poset of parts and we saw in the computational aspect that the poset of parts can be refined into a cell complex. Each cell has local coordinates and any point in a cell can be referred to using the local coordinates. Every point in the base space is in some cell, so any point in the base space can be referred to by (cell, local coordinates). We'll call this combination a cell point:

cell point = (cell, local coordinates) = (c, **u**)

where **bold face** is used to denote a vector. The cell point provides a fundamental mechanism for referring to any one of the infinite number of points in the base space, at least to within the numerical accuracy of whatever floating point type we use to represent vector components.

9.2 Local section space

The next step is to formalize the evaluation method on each piece of the base space. We assume that every cell has an associated local map:

value: $E^d \times A \rightarrow F$

where E^d is the local coordinate space for the cell and A is the parameter space for the map. A is a finite but otherwise arbitrary n-ary relation on primitive types, so $a \in A$ is an n-tuple of primitive values.

We call this type of map a parameterized computable function. We can view such a function as a family of maps:

value(*, a): $E^d \rightarrow F$

with one member of the family for each $a \in A$. This family is the local section space, the space of all maps defined by this parameterized computable function. The parameter tuple a is the data representation of a local section and the local section space is in one to one correspondence with the parameter space A. We can use A to represent the section space as a table. We'll discuss the table in more detail when we discuss the data aspect for the section role.

9.3 Global section space

We now have a local section space for each cell and the next step is to assemble all the local section spaces into a global section space. The global section space is the Cartesian product of the local section spaces. A global section is a tuple of local sections, with one component for each cell in the base space.

We can't evaluate a global section directly, we can only evaluate local sections directly. To evaluate a global section at a base space point specified by a cell point (c, **u**), we have to first select the local section corresponding to c and evaluate it at local coordinate **u**:

global.value(c, **u**) = global.restrict_to(c).value(**u**)

9.4 Compatibility conditions

The cells may overlap. If a base space point b is in more than one cell, say c and c', then it gets coordinates in each cell, (c, u) and (c', u'). The local section can be evaluated in both c and c'. But for the map to be single value we must get the same value in both cells, leading to the compatibility condition

global.restrict_to(c).value(**u**) = global.restrict_to(c').value(**u'**) wherever c, c' overlap

In terms of our local section space, this condition becomes two conditions:

1. the local evaluation methods must be compatible, and
2. the parameter tuples must be compatible.

Condition 2 selects a subset of the global section space.

9.5 Cell complexes and meshes

We assumed in the discussion above that the pieces we divided our base space into formed a cell complex. The section representation we developed however does not in fact require all the properties of a cell complex. In particular, we saw previously that a finite cell complex contains the complete dimensional cascade of cells. If d is the highest dimension in the complex, then cells of every dimension less than d must be present. In practice, many mesh types don't explicitly provide the entire dimensional cascade and our section representation doesn't require it. We require only that the cells entirely cover the base space and that every cell is equipped with a local coordinate system.

9.6 Additional reading

The parameterized computable function abstraction, which connects numerical representation of maps to relations, is a central part of the sheaf data model and is not to the best of my knowledge discussed anywhere else except:

(Butler, 2012) attempts to rigorously specify the mathematics of the sheaf data model.

However, the general style of map representation discussed in this section can be viewed as a generalization of the finite element representation of maps, so the reader may find it useful to review the finite element method.

(Hughes, 2000) is a standard text on the finite element method.

10 Data aspect for the base space role

The data aspect focuses on how to represent a base space as persistent data. Our conceptual model for a base space is a topological space with a poset of parts, which we refined in the computational aspect into a poset of cells. The central notion in the data aspect is the lattice ordered relation.

10.1 Lattice ordered relation

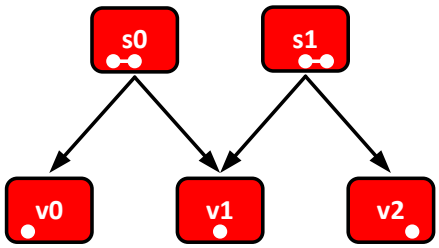
Remember that an n-ary relation is a set of n-tuples and that a poset consists of two components, an ordinary set called the base set and an order relation on the base set. We define a partially ordered relation as a poset in which the base set is itself a relation and the order relation is an arbitrary, externally specified order relation. In particular, the order relation is not necessarily derived from the attributes of the base relation.

We assume that the relation is finite. Hence the poset is finite and every finite poset has an associated finite distributive lattice. We can interpret our partially ordered relation as either "poset ordered" or "lattice ordered". The poset ordered relation refers to just the tuples and covering relation explicitly enumerated in the specification of a given instance. The lattice ordered relation interprets the explicitly enumerated members as JIMS and implicitly includes the JRMs of the associated finite distributive lattice. The lattice interpretation is more general, it includes the poset interpretation as a subset, so we will usually think of our partially ordered relation as lattice ordered.

We introduce the use of lattice ordered relations to represent base spaces by an example. As usual, the example has to be small so we can create all the diagrams. The polyline of Figure 15 will do; we repeat the covering relation graph in Figure 22 for convenient reference. We begin by introducing a relation to describe the attributes of cells:

cell: (id: string, d: int, cell type: string)

Now we can describe the basic parts in the polyline example as an instance of the cell relation, as shown in Table 7.



id	d	cell type
v0	0	vertex
s0	1	segment
v1	0	vertex
s1	1	segment
v2	0	vertex

Figure 22: Polyline graph **Table 7: Polyline cell relation**

We can now merge the two into a partially ordered relation, a poset in which each member is a tuple in a relation, as shown in Figure 23.

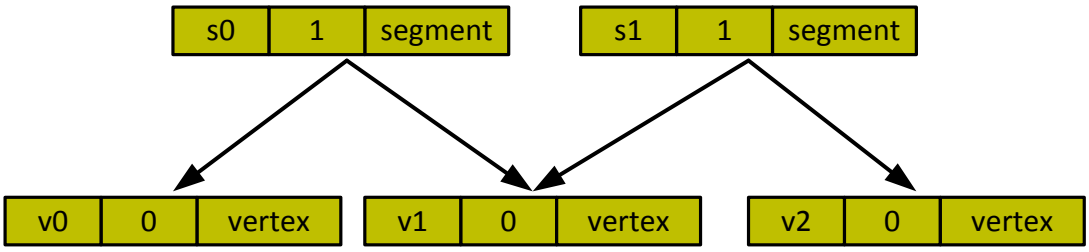


Figure 23: Polyline as poset ordered relation.

The visual representation in Figure 23 is the most accurate way to visualize a partially ordered relation. However it is not very practical for several reasons. If there are more than a few members or more than 2 or 3 attributes, the diagram can be difficult to draw. Furthermore, as we shall shortly see, we also want to associate a partial order with the columns of the relation, which is essentially impossible when drawn this way.

Since the nodes in the graph correspond to rows in the table, we usually depict an ordered relation with the graph drawn sideways, greater members are left of lesser members, and "attached" to the rows, as in Figure 24. As shown in the figure, we also associate an id with each node, rather than an attribute of the relation. This is partly just visual convention, but it is also true that the Sheaf System assigns an id to each node, so there is no need to declare an id attribute.

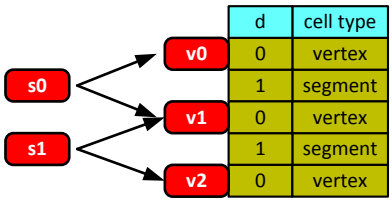


Figure 24: Polyline table with row graph "attached" to the rows.

We can represent any cell poset, in fact any partially ordered structure, and its associated finite distributive lattice as a lattice ordered relation. The general rules are:

1. The structure is represented by a lattice ordered relation with a schema appropriate to the basic parts in the structure.

2. Each basic part in the structure is represented by a JIM in the lattice with a corresponding node in the graph and a row in the table.The row contains precisely the new information the JIM introduces into the lattice. Conversely, each row in the table represents a basic part in the structure.

3. Each composite part in the structure is represented by a JRM in the lattice with a correspond node in the graph but no row in the table. A JRM corresponds to a collection of rows. In the figures, JRMs are given empty placeholder rows just to make nodes in the row graph line up with rows in the table.

Incidentally, the last rule explains why the Sheaf System provides ids for the nodes in the graph rather than relying on attributes in the relation - we have to have ids for the nodes that don't have corresponding rows in the table. Figure 25 shows the lattice ordered relation for the well of Figure 14.

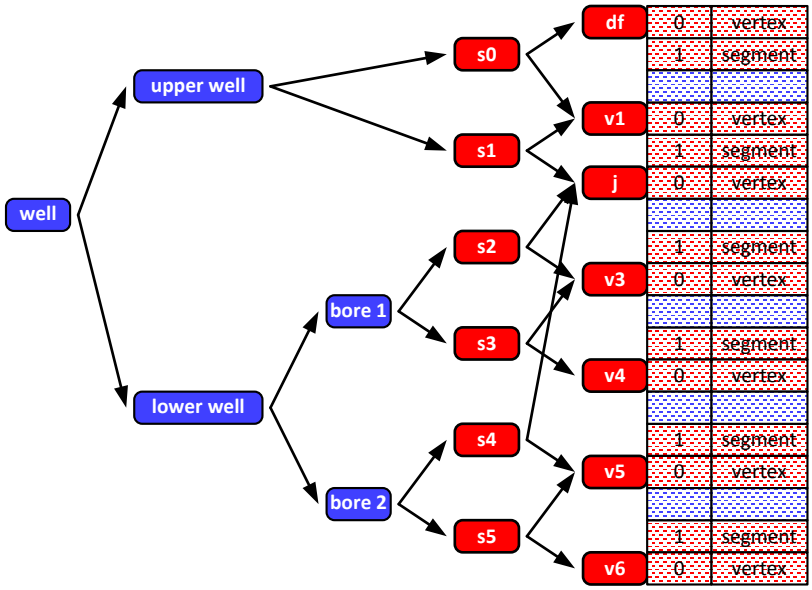


Figure 25: Well as lattice ordered relation.

10.2 Additional reading

The lattice ordered relation abstraction, like the parameterized computable function abstraction of the previous section, is unique to the sheaf data model.

(Butler, 2012) is the only reference.

But take away the partial ordering and you have an ordinary relation, as in the relational data model. The reader may find it useful to review relational database theory. There is a huge and constantly changing market for data base books. Here's two I'm familiar with.

(Date, 2003) The comprehensive, long standing classic in the field, now in its 8th edition.

(Maier, 1983) Emphasizes the mathematical theory of the relational data model. No longer in print, but available used.

11 Data aspect for the fiber space role

We saw in the discussion of the conceptual role that any type could in principle play the fiber space role but that in practice the algebraic types of mathematical physics were particularly important. In the discussion of the computational aspect, we found that these types could be represented by classes in object-oriented programming languages.

11.1 Schema lattice

As is typical of classes in object-oriented programming, the classes we are interested in are connected to each other by inheritance (is_a) and aggregation (has_a) relationships. We saw in the Part Space tutorial that these relationships can be described by a subobject hierarchy. Every object of a given class has a base class subobject associated with each direct inheritance relationship and a data member subobject associated with each direct aggregation relationship. We further saw that the subobject hierarchy can be described as a part space, that is, a finite distributive lattice.

We thus arrived at the notion of a schema lattice for a collection of classes. We gave a general set of rules for interpreting a finite distributive lattice as a class schema. We restate the rules here, explicitly in the language of finite distributive lattices:

1. A type is represented by a table with a schema lattice describing the columns of the table.

2. A schema lattice corresponds to a collection of types related by subobject inclusion and each member of the lattice corresponds to a type. We will refer to the lattice member corresponding to a type as the class schema member for the type. A typical class schema member, in practice, is somewhere in the interior of a schema lattice, that is the type both includes subobjects and is included as a subobject.

3. The JIMs of a schema lattice correspond to explicitly specified class or primitive types. This captures the notion that an explicitly specified type is more than the sum of its subobjects, it has associated operations or member functions that are implied by the type but not explicitly represented in the lattice.

4. The JRMs of a schema lattice correspond to types that can be implicitly generated from the explicitly specified types. An implicit type corresponds to a C++ struct with data members but no member functions. The data members are precisely those defined by its subobject schema (see item 7 below).

5. The atoms of a lattice schema are types with no subobjects. A atom thus represents either a class type with no base class and no data members or it represents a primitive type. This captures the notion that the parts and implementation of a primitive type are hidden and not represented in the part space.

6. The schema lattice for a given type is the down set of the class schema member. The down set of a member of a lattice is itself a (sub)lattice.

7. The subobject schema for a given type is specified by the *largest* JIMs in the strict down set of the class schema member. (The "strict" down set of a given member is the set of members strictly below the given member; it doesn't include the given member itself.). Equivalently, we can define the subobject schema to be the join of the largest basic parts. This is a unique JRM in lattice. We can thus think of the subobject schema as either an individual lattice member or a set of subobjects.

8. The relation schema for a given type is specified by the *smallest* JIMs (atoms) of primitive type in the down set of the class schema member. Equivalently, the relation schema is the join of the atoms of primitive type, which is another unique JRM. So we can also think of the relation schema as either an individual member of the lattice or a set of primitive subobjects.

9. The relation schema for a type specifies the columns in the table representing the type.

As an example, we can construct the schema lattice for the class hierarchy corresponding to the physical property type hierarchy of Figure 9. As we noted in section 8, the class hierarchy mirrors the type hierarchy. We'll focus on the schema for class E2 in order to make the diagram simpler, Figure 26. Since type inheritance implies subobject inclusion and since the cover relation arrows point in the direction opposite inclusion, we see that Figure 26 is essentially the E2 subgraph of Figure 9 turned upside down.

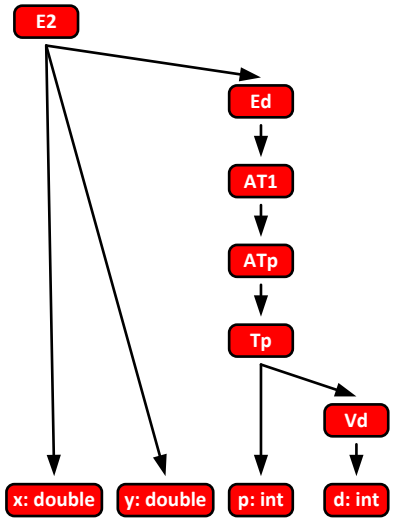


Figure 26: Schema lattice for class E2, the 2D Euclidean vector.

Figure 27 shows the subobject schema as specified by rule 7 while Figure 28 shows the relation schema as specified by rule 8.

11.2 Lattice ordered relation with schema lattice

We can combine the schema lattice abstraction presented in the preceding section with the lattice ordered relation abstraction of section 10.1. We can visualize the lattice schema as attached to the columns of a table instantiated on the relation schema. The rows of the table can be ordered independently of the columns, so we can also attach a lattice to the rows, as shown in Figure 29. The rows in this figure, each a 2D vector, do not include each other, so there are no links between the JIMs in the row graph. The "unit vectors" member is included as an example of a JRM.

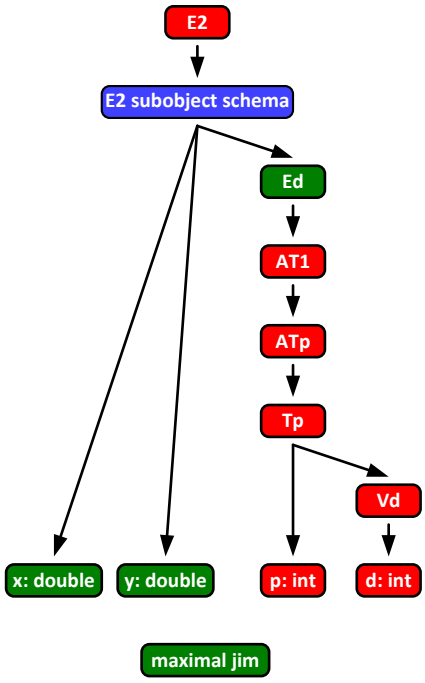


Figure 27: Subobject schema

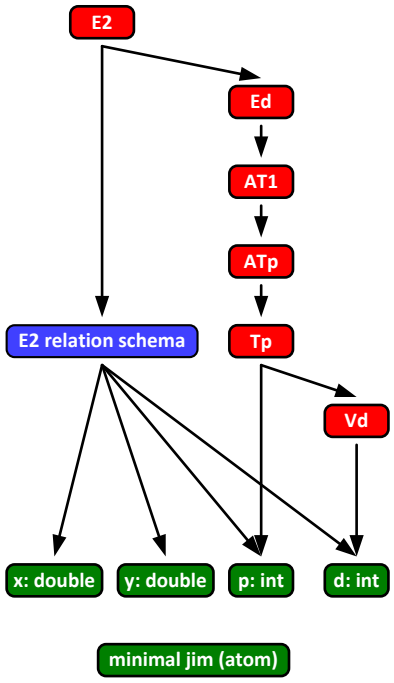


Figure 28: Relation schema

11.3 Generic class features

The lattice schema construction can be viewed as defining a generic class with the following features:

- a class schema,
- a subobject schema,
- a relation schema,
- data members specified by the schema, and
- member functions unique to the class but not specified by the schema.

To use the relation schema to support data persistence, it's reasonable to require that the class have 3 additional tuple related features:

- conversion to tuple: an operator that converts an instance of the class to a tuple specified by the relation schema;
- constructor from tuple: a constructor that takes a tuple specified by the relation schema and creates an instance of the class; and
- assignment from tuple: an assignment operator that initializes an instance from a tuple specified by the relation schema.

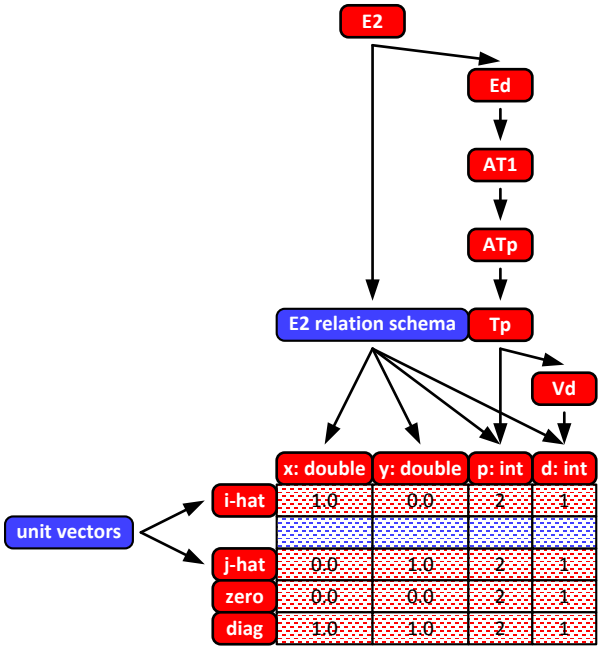


Figure 29: E2 table with lattice schema.

11.4 Subtypes and restriction

If we have a schema lattice $\mathbb{S}$ and two schema members s, s' $\in \mathbb{S}$ with s' $\leq$ s, we say the type defined by s' is a subtype of the type defined by s. We have to immediately point out that this definition of subtype is the opposite of the definition frequently used in object-oriented languages, where a derived class is usually called a subtype of its base class. However the definition given here is the natural one for a schema lattice. It has the desirable property that a subobject is associated with a subtype. The usual object-oriented definition associates a subobject with a super type.

A projection operation in relation theory selects a specified subset of the components in a tuple. Given a row r from a table T_s with schema s, we can create a subobject r' of type s' by projecting r onto the relation schema of s'. We say that r' is the restriction of row r to s'.

If we project every row in T_s onto s', we create another table $T_{s'}$. We call $T_{s'}$ the restriction of table T_s to s'. Restriction defines a map:

$\rho_{s's}: T_s \rightarrow T_{s'}$

that takes each row in T_s to its restriction in $T_{s'}$. From the definition of projection, the restriction map $\rho_{s's}$ is defined if and only if s' $\leq$ s.

If s'' is another member of $\mathbb{S}$ with s'' $\leq$ s' $\leq$ s, then from the properties of projection we have:

$\rho_{s"s} = \rho_{s"s'} \circ \rho_{s's}$ and
ρ_{ss} = identity map

Furthermore, we consider each restriction distinct, $T_s = T_{s'}$ if and only if s = s'.

If we now restrict T_s to every $s' \leq s$, we get a family $\mathfrak{I} = \{T_{s'}: s' \leq s\}$ of tables related by restriction. We can define the restriction relation $\preccurlyeq$ on $\mathfrak{I}$ by $T_s \preccurlyeq T_{s'}$ if and only if there is a restriction map $\rho_{s's}$, that is, if and only if $s' \leq s$. Note that the order in $\mathfrak{I}$ is the reverse of the order in $\mathfrak{S}$. From the properties of the restriction map given above, we have:

$T_{s"} \preccurlyeq T_{s'}$ and $T_{s'} \preccurlyeq T_s$ implies $T_{s"} \preccurlyeq T_s$,
$T_s \preccurlyeq T_s$, and
$T_s \preccurlyeq T_{s'}$ and $T_{s'} \preccurlyeq T_s$ implies $T_s = T_{s'}$

so $\preccurlyeq$ is an order relation on $\mathfrak{I}$ and $(\mathfrak{I}, \preccurlyeq)$ is a partially ordered set.

We can visualize the family of tables created by restriction using class E2. But we have to make the schema for E2 even simpler in order to make the diagram drawable. So we discard most of the E2 schema to create the simplified E2 schema and table in Figure 30. The corresponding family of table restrictions in shown in Figure 31.

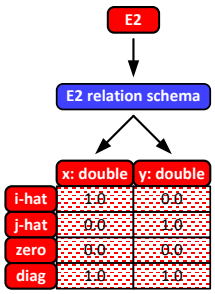


Figure 30: Simplified E2 schema and table

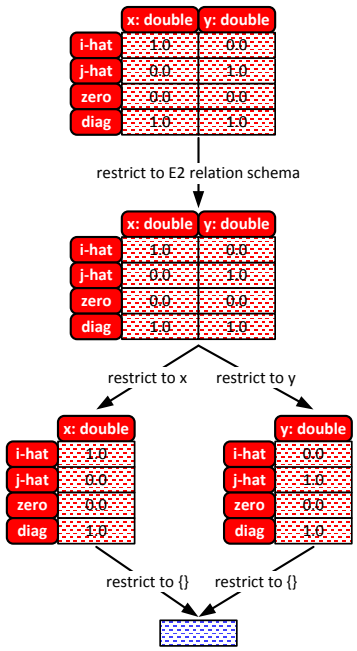


Figure 31: Family of restrictions of simplified E2

11.5 Sheaf of lattice ordered relations

The mathematical concept that ties all these ideas together and forms the central structure of the sheaf data model is the sheaf of lattice ordered relations. The sheaf concept was first introduced during and just after World War II in the context of topology as a certain kind of map from the family of open sets of a topological space to a family of sets of some specific type, for instance groups. Many treatments of sheaf theory still address this notion of sheaf. But the family of open sets of a topological space forms a special kind of lattice called a complete Heyting algebra and it was soon recognized that the definition of a sheaf relied only on this lattice structure, not the fact that the members of the lattice were the open sets of a topological space. So the sheaf concept was redefined in terms of Heyting algebra and from there generalized still further in terms of category theory. Most modern treatments of sheaf theory address the categorical version.

Fortunately, we need only the simplest aspects of sheaf theory. It turns out that every finite distributive lattice is also a complete Heyting algebra, so for our limited needs, we can define the sheaf concept in terms of the finite distributive lattice. For our purposes, a sheaf is an order reversing map ϕ from a finite distributive lattice $\mathfrak{S}$ to a family of sets $\mathfrak{I}$ partially ordered by restriction. Order reversing means that for all s, s' $\in \mathfrak{S}$, s' $\leq$ s implies f(s) $\preccurlyeq$ f(s').

Our specific interest is when $\mathfrak{S}$ is a schema lattice for a lattice ordered relation and $\mathfrak{I}$ is the corresponding family of table restrictions. We thus have a sheaf of lattice ordered relations.

The sheaf for our simplified E2 example is shown in Figure 32.

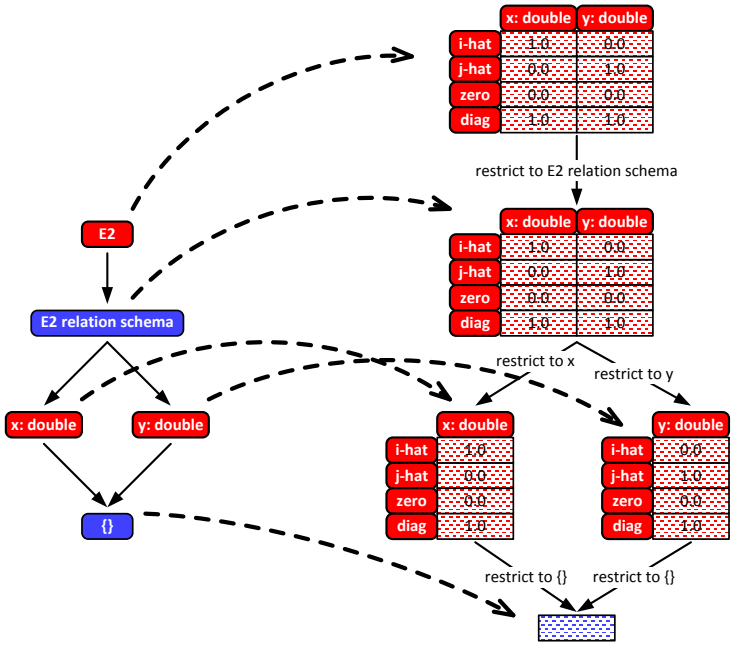


Figure 32: Sheaf of lattice ordered relations for simplified E2

11.6 The sheaf data model

The sheaf data model interprets every type as a sheaf of lattice ordered relations, represented by a table combined with a row graph and a column graph. We refer to this combination as a sheaf table. A sheaf data base is a collection of sheaf tables. Every table has an associated table, its schema table. The row lattice of the schema table defines the schema lattice of the former table, which we will refer to as the object table. The schema recursion terminates in a special table, the primitive schema table, which is its own schema table. (The primitive schema table was called the schema part table in the Part Space and Part Spaces for Scientific Computing tutorials.) Every sheaf database contains two additional special tables. The primitives table is a special schema table that describes the memory requirements for each primitive type supported by the system. Finally, the namespace table is a table of contents for a sheaf data base. It contains a basic part for each other table in the data base.

11.7 Additional reading

The schema lattice concept is another sheaf data model specific concept.

(Butler, 2012) is the only reference.

But the reader may want to review lattice theory while reading this section.

(Davey, et al., 2002) is the one to have handy.

Sheaf theory has traditionally been firmly in the domain of pure mathematics and only within the last decade or so begun to find its way into applications. As a result, there is no good introductory text on sheaf theory.

(Goldblatt, 1984) is mostly about category theory and logic, but section 4.5 "Bundles and sheaves" and section 14.1 "Stacks and sheaves" are fairly readable, if incomplete, and actually draw a picture or two.

(Tennison, 1976) treats the topological space version of sheaves, mostly from the point of view of set theory rather than category theory, which is helpful. Chapters 1 and 2 are relevant but slow going.

(Miraglia, 2006) is a comprehensive reference and understandable, if you read very carefully. Chapters 22 and 23 are the most relevant. His "discrete sheaf" in example 23.11 is something very close to our definition of sheaf. Pretty dense material, over 500 pages and not a single picture, unless you count the arrow chasing diagrams of category theory.

(Mac Lane, et al., 1992) is another well known reference. Chapter 2 is the most germane. But again, pretty tough sledding. This one has 600 pages without any pictures except arrow chasing.

12 Data aspect for the section role

The discussion of the computational aspect for the section role developed the parameterized computable function as the local section space associated with each cell in the base space. The global section space was represented by the collection of local section spaces. The task of this section is to describe the data representation of a section. As with all objects in the sheaf data model, a section will be represented as a row in a table, so the task becomes defining the schema of the table.

We previously developed the lattice ordered relation to describe the data aspect of base spaces and developed the schema lattice to describe the data aspect of fiber spaces. We then combined the two structures into a sheaf of lattice ordered relations which can represent either a base space or a fiber space. The major motivation for doing so is that we need to combine the two. The schema for a section space is a combination of the row lattice of its base space with the schema lattice of its fiber space. The sheaf abstraction gives us the mathematical space in which to construct this combination.

In the following, let B be a topological space with a cell complex C, let *C* be the poset with base set C, ordered by point set inclusion, and let $\mathcal{C}$ be the finite distributive lattice associated with *C*. Let F be an arbitrary property space with schema poset *F* and schema lattice $\mathcal{F}$. Let *S* be the schema poset and $\mathcal{S}$ be the schema lattice for a section space S with base space B and fiber space F.

12.1 Local section class

We can represent the local section space (parameterized computable function) abstraction as a class. Recall that the generic class associated with a schema lattice, described in section 11.3, has the following features:

- class, subobject, and relation schema
- class specific member functions, and
- tuple conversion, constructor, and assignment functions.

A local section class has two additional member functions:

- value_at_coordinate: **U** $\rightarrow$ F, where **U** is the local coordinate space for the section and F is the fiber space for the section, computes the value of a section, and
- coordinate_at_value: F $\rightarrow$ **U** computes the inverse, if it exists.

A local section schema is the class schema for a local section class. Since the additional member functions of a local section class are not represented in a schema lattice anyway, a local section schema can be any general schema lattice. The only constraint is implicit - the associated class has to have the value_at_coordinate and coordinate_at_value member functions. In principle, the fiber space is not specified by the schema, only by the signature of these functions. We will see shortly that in practice the schema usually does specify the fiber space.

It is worth emphasizing that a local section *class* represents a local section *space*. The relation schema of the class defines the parameter space of the parameterized computable function. As described previously, each tuple in the parameter space corresponds to an individual section. The data members of an instance of a local section class correspond to a tuple of the parameter space, so each instance of a local section class corresponds to a local section. The relation tuple associated with the local section object is the data representation of the section.

12.2 Global section class

A global section schema is a general lattice schema that has one or more local section schema as subtypes. A global section schema class is a general class with a single additional member function:

- local: $C \rightarrow \mathbb{S}$ takes a cell id as input and returns the id of the local section schema member for the cell as output.

A global section class is a general class with a schema which is a global section schema. A global section class has 3 additional member functions:

- restrict_to: $C \rightarrow$ local section, restricts the global section to the local section over a specified cell,
- value_at_cell_point: $C \times \mathbf{U} \rightarrow F$ computes the value of the section at a cell point, and
- cell_point_at_value: $F \rightarrow C \times \mathbf{U}$ computes the inverse, if it exists.

The global section class represents the global section space. In particular, the relation schema defined by the schema of the class defines the global parameter space. An instance of a global class represents a global section and the associated relation tuple defines the data representation of the section.

12.3 Examples

One can construct a global section schema that meets the above requirements in very many different ways. We first show two examples that represent extreme cases, then describe a construction that occurs more frequently in practice.

The simplest global section schema is constructed by associating all the cells in the base space with the same local schema. This implies that the section has a constant value everywhere on the base space. A simple example is shown in Figure 33.

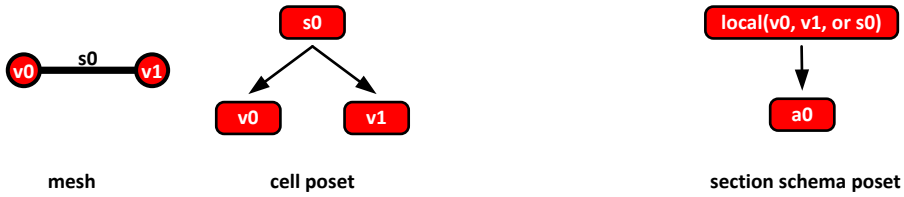


Figure 33: Constant schema.

At the opposite extreme, the most complex global section schema is constructed by associating each cell with a different, arbitrary local section schema. The global schema is the sum (disjoint union) of local schema for each cell, as in Figure 34. Of course, ensuring the compatibility conditions can be tricky with such a schema.

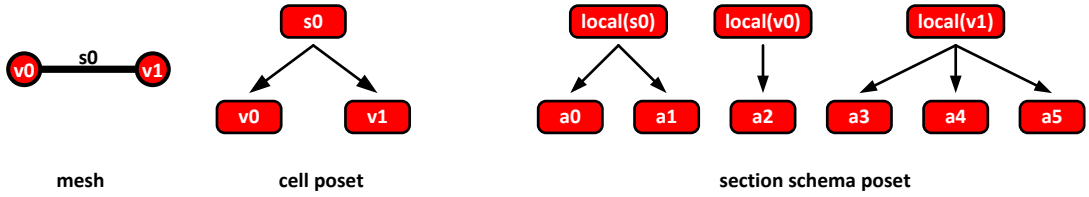


Figure 34: Sum schema.

Neither of these constructions is common in practice. As we discussed in Part Spaces for Scientific Computing, it is extremely desireable in practice to be able to restrict a section to any part of the base space or fiber. This corresponds to constructing the section schema poset as the Cartesian product of the cell poset and the fiber schema poset, as shown in Figure 35.

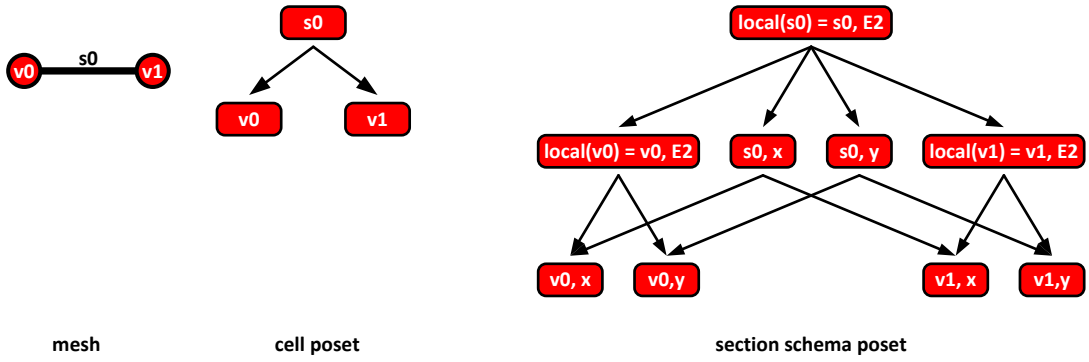


Figure 35: Product schema.

12.4 Lattice tensor product

We saw in section 7.1 that the Birkhoff representation theorem says that associated with every finite poset P there is a finite distributive lattice $\mathcal{L} \sim \mathcal{O}(P)$. If $P = A \times B$ is a Cartesian product, the associated finite distributive lattice is the tensor product of the

lattice $\mathfrak{A} = \mathcal{O}(A)$ and the lattice $\mathfrak{B} = \mathcal{O}(B)$. We'll denote the tensor product of $\mathfrak{A}$ and $\mathfrak{B}$ by $\mathfrak{A} \otimes \mathfrak{B}$, so in symbols:

$\mathfrak{A} \otimes \mathfrak{B} = \mathcal{O}(J(\mathfrak{A}) \times J(\mathfrak{B}))$ or $\mathcal{O}(A) \otimes \mathcal{O}(B) = \mathcal{O}(A \times B)$

The members of the tensor product lattice are joins of pairs:

$(a_1, b_1) \vee (a_2, b_2) \vee \ldots \vee (a_n, b_n)$

where $a_i \in \mathfrak{A}$, $b_i \in \mathfrak{B}$ and $a_i \neq a_j$ and $b_i \neq b_j$ for any i and j.

12.5 Hetergenous section representation

For a section schema, if the section space schema poset $S = C \times F$, then the schema lattice $\mathfrak{S}$ is the tensor product $\mathfrak{S} = \mathfrak{C} \otimes \mathfrak{F} = \mathcal{O}(C \times F)$. The members of the section space lattice are joins of pairs (base space part, fiber schema part). A section instantiated on such a schema has different fibers on different parts of the base space. If we take this one step further and make the fiber schema the sum of different representations we get a heterogeneous section representation in which fiber space is the same everywhere, but the representation of the section varies from part to part in the base space. For instance, given the base space lattice and fiber space schema lattice shown in Figure 36, we can choose a section member given by the join (upper well, CartesianE2) $\vee$ (lower well, PolarE2). A section instantiated on this schema will use a Cartesian representation on the upper well and a polar representation on the lower well. The junction will be represented twice, so the compatibility condition must be enforced in some way.

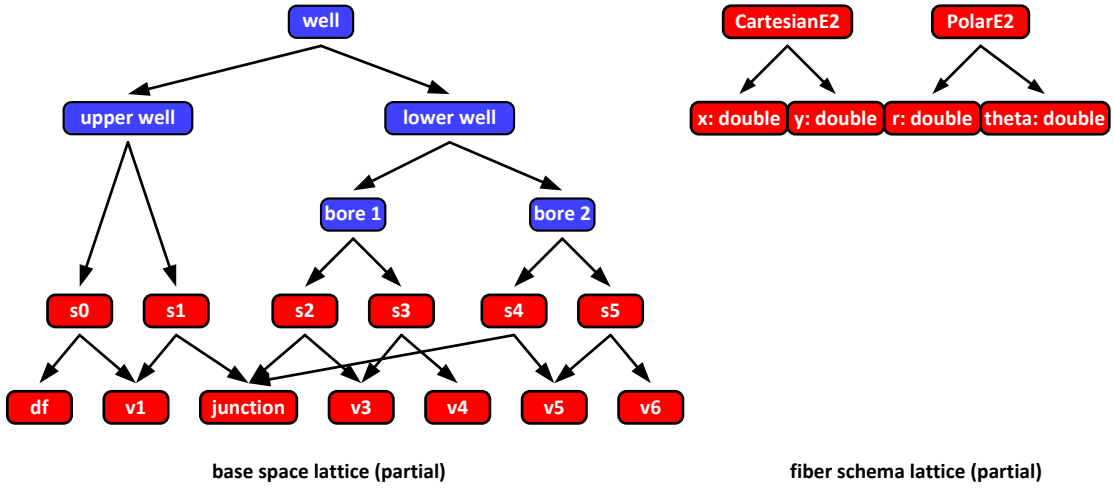


Figure 36: Heterogeneous section representation.

12.6 Additional reading

There is no good introductory treatment of the tensor product for finite distributive lattices that I am aware of. The tensor product for general lattices has several definitions

in the literature; most of the treatments are based on category theory. None take advantage of the simplifications possible for finite distributive lattices.

(Gratzer, 2006) defines the tensor product for finite (but not necessarily distributive) lattices, but the definition given there depends on the more abstract notion of a free lattice.

(Shmuely, 1979) defines the tensor product for general distributive lattices. Theorem 1.4 constitutes a really obscure way of stating the definition given in this document.

Appendix A References

Abraham, Ralph, Marsden, Jerrold E. and Ratiu, Tudor. 1983. *Manifolds, Tensor Analysis, and Applications.* Reading : Addison-Wesley, 1983.

Birkhoff, Garrett. 1995. *Lattice Theory.* Third. Providence : American Mathematical Society, 1995.

Bishop, Richard L. and Goldberg, Samuel I. 1980. *Tensor Analysis on Manifolds.* New Yrok : Dover Publications, 1980.

Butler, D. M. 2001 (revised 2012). Sheaf Data Model: Context and Overview. *Limit Point Systems, Inc.* [Online] 2001 (revised 2012). http://www.sheafsystem.com/images/Publications/sdm_context_and_overview.pdf.

Butler, David M. 2012. The Sheaf Data Model, Part 1: Objects. *Limit Point Systems, Inc.* [Online] 2012. http://www.limitpoint.com/images/Publications/The%20Sheaf%20Data%20Model.pdf.

Crampin, M. and Pirani, F.A.E. 1986. *Applicable Differential Geometry.* Cambridge : Cambridge University Press, 1986.

Date, C. J. 2003. *An Introduction to Database Systems.* Reading : Addison-Wesley, 2003.

Davey, B. A. and Priestley, H. A. 2002. *Introduction to Lattices and Order.* Second Edition. Cambridge : Cambridge University Press, 2002. ISBN 0-521-78451-4.

Fuks, D. B. and Rokhlin, V. A. 1984. *Beginner's Course in Topology: Geometric Chapters.* Berlin : Springer-Verlag, 1984.

Goldblatt, Robert. 1984. *Topoi: The Categorical Analysis of Logic.* Amsterdam : North-Holland, 1984.

Gratzer, G. 2006. *The Congruences of a Finite Lattice, A Proof-by-Picture Approach.* Boston : Birkhuaser, 2006.

Halmos, Paul R. 1974. *Naive Set Theory.* New York : Springer-Verlag, 1974.

Hughes, Thomas J.R. 2000. *The Finite Element Method.* New York : Dover Publications, 2000.

Janich, Klaus. 1994. *Linear Algebra.* New York : Springer-Verlag, 1994.

—. 1984. *Topology.* New York : Springer-Verlag, 1984.

Koenderink, Jan J. 1990. *Solid Shape.* Cambridge : MIT Press, 1990.

Mac Lane, Saunders and Moerdijk, Ieke. 1992. *Sheaves in Geometry and Logic, A First Introduction to Topos Theory.* New York : Springer-Verlag, 1992.

Maier, David. 1983. *The Theory of Relational Databases.* Rockville : Computer Science Press, 1983.

Miraglia, Francisco. 2006. *An Introduction to Partially Ordered Structures and Sheaves.* Milan : Polimetrica, 2006.

Munkres, James R. 1975. *Topology: A First Course.* Englewood Cliffs : Prentice-Hall, 1975.

Nash, Charles and Sen, Siddhartha. 1983. *Topology and Geometry for Physicists.* London : Academic Press, 1983.

Rosen, Kenneth H. 1995. *Discrete Mathematics and Its Applications.* New York : McGraw-Hill, 1995.

Shmuely, Zahava. 1979. The tensor product of distributive lattices. *Algebra Universalis.* 1979, Vol. 9, pp. 281-296.

Tennison, B. R. 1976. *Sheaf Theory.* Cambridge : Cambridge University Press, 1976.