2003

A Mathematical Model for Simplifying Representations of Objects in a Geographic Information System

Gabriel Perrow

A MATHEMATICAL MODEL FOR SIMPLIFYING REPRESENTATIONS OF OBJECTS IN A GEOGRAPHIC INFORMATION SYSTEM

By

Gabriel M. Perrow

B.A. University of Maine, 2001

A THESIS

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Master of Arts

(in Mathematics)

The Graduate School

The University of Maine

August, 2003

A MATHEMATICAL MODEL FOR SIMPLIFYING REPRESENTATIONS OF OBJECTS IN A GEOGRAPHIC INFORMATION SYSTEM

By Gabriel M. Perrow

Thesis Advisor: Dr. Robert Franzosa

An Abstract of the Thesis Presented
in Partial Fulfillment of the Requirements for the
Degree of Master of Arts
(in Mathematics)
August, 2003

The study of operations on representations of objects is well documented in the realm of spatial engineering. However, the mathematical structure and formal proof of these operational phenomena are not thoroughly explored. Other works have often focused on query-based models that seek to order classes and instances of objects in the form of semantic hierarchies or graphs. In some models, nodes of graphs represent objects and are connected by edges that represent different types of coarsening operators.

This work, however, studies how the coarsening operator "simplification" can manipulate partitions of finite sets, independent from objects and their attributes. Partitions that are "simplified" first have a collection of elements filtered

(removed), and then the remaining partition is amalgamated (some sub-collections are unified).

Simplification has many interesting mathematical properties. A finite composition of simplifications can also be accomplished with some single simplification. Also, if one partition is a simplification of the other, the simplified partition is defined to be less than the other partition according to the simp relation. This relation is shown to be a partial-order relation based on simplification. Collections of partitions can not only be proven to have a partial-order structure, but also have a lattice structure and are complete.

In regard to a geographic information system (GIS), partitions related to subsets of attribute domains for objects are called views. Objects belong to different views based whether or not their attribute values lie in the underlying view domain. Given a particular view, objects with their attribute n-tuple codings contained in the view are part of the actualization set on views, and objects are labeled according to the particular subset of the view in which their coding lies.

Though the scope of the work does not mainly focus on queries related directly to geographic objects, it provides verification for the existence of particular views in a system with this underlying structure. Given a finite attribute domain, one can say with mathematical certainty that different views of objects are partially ordered by simplification, and every collection of views has a greatest lower bound and least upper bound, which provides the validity for exploring queries in this regard.

ACKNOWLEDGMENTS

First, I would like to thank my advisor, Dr. Robert Franzosa, for his patience and guidance throughout the writing of my thesis. His encouraging words and persistence are appreciated.

I would also like to thank Dr. Ali Ozluk and Dr. Sundar Subramanian for their assistance on my committee, as well as their instruction and friendship while attending the University of Maine.

I wish to extend a sincere thanks to Dr. Chip Snyder for all of his encouragement and guidance throughout my education. He is a kind friend and his passion for helping others is highly admired.

My grateful appreciation goes to all of my colleagues in the graduate office, and a special thanks to Rebecca Rozario, Dan Juska, Todd Zoroya, and Rich Beveridge for their true friendship, generousity, and many excellent discussions in regard to mathematics and life.

Thank you to my father, Mark Perrow, for keeping me focused on my dreams and not letting me settle for anything less than what I am capable of. Also, thanks to my wife Carybrooke for being so supportive and making sacrifices with her time and energy to help me write this thesis.

Partial funding for this research was provided by National Imagery and Mapping Agency under grant number NMA201-00-1-2009, and is appreciated.

Finally, I thank the rest of my family, in-laws, and friends for all of the love and support they've given to help me write this thesis, and encouraging me to succeed in all my life endeavors.

TABLE OF CONTENTS

ACKNOWLEDGMENTS....................ii

LIST OF FIGURES....................v

Chapter

1. INTRODUCTION....................1
2. BASIC G.I.S. TERMINOLOGY....................4
    2.1 Objects and Attributes....................4
    2.2 Attribute Domains....................6
    2.3 Object Codings....................7
    2.4 Summary....................8
3. PARTITIONS....................9
    3.1 Defining Partitions....................9
    3.2 Basic Operations on Partitions....................11
        3.2.1 Filter/Insert....................11
        3.2.2 Amalgamation/Refinement....................14
    3.3 Simplification/Complexification....................18
    3.4 Summary....................21
4. ORDER RELATIONSHIPS BASED ON SIMPLIFICATION....................23
    4.1 Basic Lemmas....................23
    4.2 Composition Theorem....................27
    4.3 Partial-Order Structure....................30
    4.4 Summary....................34

5. SIMPLIFICATION LATTICES OF PARTITIONS........35

5.1 Background........35

5.2 Key Lemmas........36

5.3 Simplification Lattice Proof........43

5.4 Summary........46

6. VIEWS AND REPRESENTATIONS........47

6.1 Views........47

6.2 Actualization of Views........48

6.3 Example........49

6.4 Summary........53

7. CONCLUSIONS AND FURTHER WORK........55

BIBLIOGRAPHY........57

BIOGRAPHY OF THE AUTHOR........58

LIST OF FIGURES

Figure 3.1 A partition of the integer set [0, 10] 10

Figure 3.2 Filter and insert serve as inverse operations on partitions by removing or adding in collections of sets 14

Figure 3.3 Amalgamation unifies sub-collections of a partition, while refinement partitions elements into a new single partition 17

Figure 3.4 Simplification and complexification are inverse operations of one another, where simplification is filter followed by amalgamation, and complexification is refinement followed by insert 21

Figure 4.1 The composition of two or more simplifications can be accomplished in one step with a single simplification 30

Figure 5.1 Given that P and P' partition the same set, note that $P_j \in P$ belongs to many subsets sub-partitioned by both P and P', but I is the intersection of all such sets 40

Figure 5.2 The simplification lattice of the three point set {a, b, c} 45

Figure 6.1 A representation of objects is based on the view actualization, which depends on which objects have codings contained in various partitioned subsets of the view 51

Figure 6.2 The representation shows the result of a simplification of the view V 53

Chapter 1

INTRODUCTION

A computer system capable of assembling, storing, manipulating, and displaying geographically referenced data is called a Geographic Information System (GIS). The data is often used to study challenging planning and management issues and to try to generate solutions. A user analyzes data in the GIS by asking queries and selecting the appropriate results. Data is often represented visually and has direct application to real world geo-spatial scenarios. Representations like maps or graphs can be generated by data in the system, and can be manipulated to reflect the desired goals of someone studying the information.

One such goal is to have the ability to study the data representations according to different levels of detail. If a person desires to find out information on a topic, he or she may want a more generalized representation of the data then they are given. Depending on the query, a person may want to see more of the general trends or patterns of a geographic region, or conversely, to view that same region with more specific semantics in mind.

For example, think about a map showing all of the mountains higher than 1000 ft in the state of Maine. We may not be interested in every individual mountain's name and location that satisfies this condition, but we might desire to see groupings of different regions. There could exist a number of individual mountains in the Acadia and Katahdin regions, but we might group all the mountains regionally together by showing them as two symbols labeled, the "Acadia Region" and the "Katahdin Region".

The resulting representation would be termed has having lower "granularity", or level of detail, and would be a semantic generalization of the individual mountains (Stell and Worboys 1999). Instances of objects (mountains), were grouped together into classes according to regional containment (Ramalingam 2002).

Thus, our main interest lies in this idea of generalization, and how we can formally capture the concept mathematically. To do this, we must create the framework in which objects (e.g. Mt. Katahdin, Mt. Cadillac) could be described according to important semantics determined by their attributes, and could be manipulated to reflect a desired generalization or refinement of their representation.

This framework would not focus on the objects themselves though, but rather on partitions of sets. Defining a system that operates on partitions of sets instead of classes and instances of objects has some mathematical advantages. Partitioned sets can be expressed more generally than objects and can be related to other areas of mathematics. Also, particular theorems and notation related to set theory are more applicable and may be used to define or describe results in this framework. Finally, defining operations on partitions will allow the creation of a structure that could validate and hopefully accommodate some queries related to a GIS.

The next section will define some basic terminology and will provide background information of topics related to a GIS. Once this information has been established, we shall attempt to develop a well-defined mathematical structure around operations on partitions, particularly the operation simplification. After this, we shall tie our initial GIS terminology together with the developed structure, and discuss the significance of our results in terms of representations of objects.

Chapter 2

BASIC G.I.S. TERMINOLOGY

2.1 Objects and Attributes

To begin, we need to start with the terms that will be frequently used in our work. We shall define these terms formally so there is no ambiguity in regard to the vocabulary. Thus, we shall first define what we mean by an *object*.

Definition of "Object"- A physical structure or phenomenon of interest that has distinct attributes. We shall use $\mathbf{O}$ to designate the universe of all objects in the GIS.

As you can see, the definition allows an object to be virtually anything that has tangible attributes. In the context of a geographic information system, examples of objects could include buildings, mountains, forests, cities, counties, states, as well as many others. Each is a physical item that has attributes like, height, shape, area, or demographics.

To make sure you know what is meant by the word *attribute*, we shall formally define this word as well.

Definition of "Attribute" - A specific type of quality or property associated to an object.

Let's say that some objects of interest in a GIS were a collection of buildings on the UM campus, and there were four attributes of interest: materials, age, exact location, and height. Thus, each object has four "attributes" in the system.

In work done by Ramalingam, objects were defined as either being physical items or categories. These were termed as "instances" or "classes" of objects. Our work differs because it only considers "instances of objects" as being objects. For example, we consider Cadillac Mountain to be an object, but the "Acadia Region" which contains Cadillac Mountain is a class or category, and therefore is not an object by our standards (Ramalingam 2002).

Thus, since objects all are instances and have the same attributes, but not necessarily the same traits in regard to those attributes, each object would have specific values for the attributes. We call these "attribute values".

Definition of "Attribute Value" – A distinct value related to an object according to a particular attribute.

For example, on the University of Maine campus, if possible attributes for buildings are "materials, age, location, and height", then Gannett Hall might have the respective attribute values

{brick materials, 30 years old , 45.3143 north latitude - 60.8964 west longitude, 52' tall}.

Thus, these attribute values are distinctly related to Gannett Hall. We call this n-tuple of values an *object-coding* of the object "Gannett Hall". We shall formally define what this means in section 2.3.

2.2 Attribute Domains

In regard to common sense attribute values, it makes sense that Gannett Hall would not have attributes: {molten rock, 1000 years old, 45i north latitude-3i east longitude, -10 feet tall}. These values are ridiculous. Thus, there is a domain of possible attribute values that each object may possess in regard to each attribute. We call each set of all possible attribute values an *attribute domain.*

Definition of "Attribute Domain" - For each attribute, there is a finite set of elements associated to it such that the set represents all possible attribute values that can be assigned to objects in the system. If there are n attributes in the system, then $A_1, \ldots, A_n$ represent the attribute domains of possible values for each of the n attributes.

Thus the attribute domain for building materials might be the set A_1 = {wood, brick, wood/brick, steel}. In regard to age, A_2 = {10, 20, 30} could be a domain of possible values in years related to the buildings in the system. Likewise, A_3, and A_4 would be possible values for longitude/latitude and height in feet respectively. Note that the order of these domains is assumed to be finite here, though it could be an advantage in some future work to explore the notion of countably or uncountably infinite attribute domains

In regard to the formal mathematics we will develop, attribute domains serve as the underlying sets that will be partitioned, so we may establish a level of separation in regard to objects and their attributes. In chapter 3, we will define what it means to *partition* a general finite set, and in chapter 6, we will look at different partitions or *views* of subsets of an attribute domain.

2.3 Object Codings

Since each object O in **O** has n specific attribute values, one from each of the n attributes, it makes sense to define a function mapping objects to their respective attribute value n-tuples. Thus, we have a means of characterizing each object with corresponding values. We shall show this relationship via the "object coding" function.

Definition of "object coding" - For an object $O \in \mathbf{O}$, there is a function f_i that maps O to a specific attribute $a_i \in A_i$, $i = \{1, 2, \ldots, n\}$. Thus, the object coding function is a function f such that each object O is mapped to the product-space of all the attribute domains. In other words, $f: \mathbf{O} \rightarrow A_1 \times A_2 \times \ldots \times A_n$, so

$f(O) = (f_1(O), f_2(O), \ldots, f_n(O))$.

Note that f is obviously a function, since it can be said that each object maps to only one ordered n-tuple. It cannot necessarily be said though that f is either one-to-one or onto. Some distinct elements could have identical n-tuples (same height, color, etc), thus making f not injective. Likewise there might exist some n-tuples of attributes in the attribute domain product space that have no object mapped to them, thus making f not surjective. Note though that if some quality like “absolute location” were included as an attribute, one could assume there was a unique mapping of objects to their absolute location and therefore achieve one-to-one correspondence in terms of the whole n-tuple.

2.4 Summary

In review, the key GIS terminology we focused on in this section were objects, attributes, attribute values and domains, and object coding functions. Each object can be characterized by an n-tuple of attribute values called an object coding. Such a coding has one value for each attribute, and values are each contained within the attribute domain.

These terms are important because they relate to the concepts that are motivating the mathematics in this paper. Ultimately, we will show how objects can be represented according their attributes and where they lie in the attribute domain. Also, we will see how to change the level of detail of representations. This level of discernment will be accomplished through the development of operations on *partitions.*

In the next section, we will define what it means to partition a set. With this notion, we will begin to develop a framework that will eventually display representations according to their attributes, and we will define operations that manipulate partitions in order to yield new partitions of lesser or greater detail.

Chapter 3

PARTITIONS

3.1 Defining Partitions

Before defining a partition, let's first look at a helpful notion that will make our work easier thoughout the paper. Since it will be a common action for us to take the union of sets, the following function will give us a shortcut for describing the union of a collection of sets. We shall call it a *u-function*.

Definition of "u-function" – Let $C = \{C_1 ,\dots, C_k \}$ be a collection of sets. Thus the u-function of C is the union of all the elements in C. We write this as

$u(C) = C_1 \cup \dots \cup C_k$

Throughout this paper, we will be manipulating finite sets, including sets of consecutive nature numbers. Thus, the notation [a,b] will represent the set

$[a , b] = \{a, a+1, \dots, b-1, b\}$, where a and b are integers.

For example if $C = \{ \{0\} , [1, 4] , [5, 6] \}$, then

$u(C) = u\ (\{ \{0\} , [1, 4] , [5, 6] \}) = \{0\} \cup [1, 4] \cup [5, 6] = [0, 6]$.

Though fairly trivial, the u-function can save us a lot of notational effort. In fact, we shall put it to use in our next definition. Let's define what it means to *partition* a set X:

Definition of "Partition" - A partition of a finite set X is a collection $\{X_1,\ldots,X_m\}$ of mutually disjoint subsets of X such that the union of all these subsets equals X, that is, $u(\{X_1,\ldots,X_m\}) = X$.

A numerical example of a partition would be the set

$P = \{ X_1, X_2, X_3, X_4\} = \{ [0, 2], [3, 4], [5, 6], [7, 10]\}$,

where P is a partition of X = [0 , 10]. Note that the sets contained in P are mutually disjoint and u(P) = X.

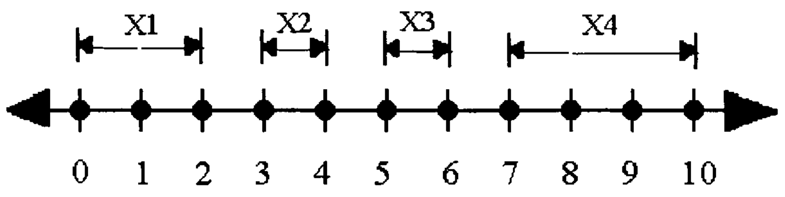


Figure 3.1 A partition of the integer set [0, 10].

Thus, partitions allow us to divide sets into collections of subsets and still preserve the underlying set. In regard to the previous section, this concept is important when we consider how to make lesser or greater distinctions between objects. If each object's coding belongs to some partitioned subset of the attribute domain, we therefore could group objects as similar or dissimilar depending on the partitioned subset containing their coding.

Of course, a partition of an attribute domain would be a partition of a product n-space, and not just a single integer interval such as [0, 10]. For our purposes though, basic 1-space sets will suffice for examples while we develop the functions and definitions we need. Later we will generalize our work to n-space attribute domains and the meaningful representations of objects based on this notion.

3.2 Basic Operations on Partitions

With partitions now defined, our next step is to create some basic types of operations on partitions that will modify or change them in desired ways. We need methods of taking a given partition and changing it in order to reflect a desired granularity. We will accomplish this task by looking at two types of operations: filter/insert and amalgamation/refinement.

3.2.1 Filter/Insert

First, we will define a basic type of operation that starts with a partition P and removes or adds a collection of sets. Since we can think of this idea metaphorically as "filtering" out or "inserting" sets into a partition, we shall call this operation type, *filter / insert.*

First, let's explicitly define what it means to *filter* a partition.

Definition of "Filter" – Given a finite set X and partition P of X, let F $\subseteq$ P. Define filt(P;F) to be the partition of X\u(F) given by P\F. We say that F is "filtered" from P.

For example, let X = [0,10] and let P = {[0, 2] , [3 , 4] , [5, 6] , [7, 10]}. If F = {[0,2] , [5, 6]} , then

filt (P ; F)

= filt ({ [0, 2] , [3 , 4] , [5, 6] , [7, 10]} ; {[0,2], [5,6])

= { [0, 2] , [3 , 4] , [5, 6] , [7, 10] }\ {[0 , 2], [5 , 6]}

= {[3 , 4] , [7, 10]}.

and filt (P ; F) is a partition of X \ u (F) = [3 ,4] ∪ [7, 10].

Notice that the removal of elements from a partition is actually a way that information can be removed from a partition <u>and</u> its underlying set. This idea is useful for when certain values in the partitioned set are no longer of interest.

In terms of broader application, filter is important because it is one way that representations of objects can have detail removed. Thus far, we have seen how the filter operation coarsens the granularity of a partition by removing elements from the collection. Thus, there is less information to see. Later, we will see how filtering partitions can lead to filtering objects.

In the work done by Stell and Worboys, filter is termed as "selection", and the operation is applied primary to graphs instead of partitions. The same idea holds though, since particular nodes and their connected paths are removed via the operation, and the result has less information. Thus, a resulting graph is generated with the absence of these details (Stell & Worboys 1999).

Along with "filter" and the removal of partitioned subsets, we shall also define "insert", which adds in sets. These two operations are paired as one type of operation because they give us the freedom to move backward and forward when manipulating partitions, and they serve as inverse operations of one another.

Formally, we define *insert* as the following:

Definition of "Insert" – Given a finite set X and a partition P' of X, let F be a collection of mutually disjoint sets such that $u(F) \cap X = \varnothing$. Define ins(P';F) to be a partition of $X \cup u(F)$ given by $P' \cup F$. We say that F is inserted into P'.

For example, let $X = [3,4] \cup [7,10]$ and let $P' = \{[3,4], [7,10]\}$. If $F = \{[0,2], [5,6]\}$, then we can say that

$$\begin{aligned} &\text{ins}(P';F) \\ &= \text{ins}(\{[3,4], [7,10]\};\{[0,2], [5,6]\}) \\ &= \{[3,4], [7,10]\} \cup \{[0,2], [5,6]\} \\ &= \{[0,2], [3,4], [5,6], [7,10]\} \end{aligned}$$

and ins(P;F) is a partition of $X \cup u(F) = [0,10]$

Notice that the partitions were chosen to conveniently show how insert acts as the inverse operation of filter. In the figure below, you can see how filter generated a partition, and how insert led back to the original.

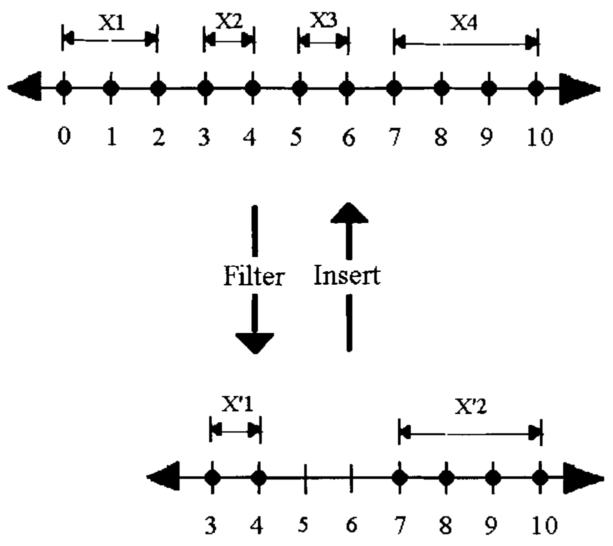


Figure 3.2 Filter and insert serve as inverse operations on partitions by removing or adding in collections of sets.

3.2.2 Amalgamation/Refinement

The other basic type of operation on partitions is amalgamation/refinement (amal/ref). Amalgamation generally means that a collection of items are combined into one. Refinement means breaking apart single entities into many. In terms of partitions, amal/ref does not behave like filter/insert, but rather serves to combine or split sets in a partition.

First, let's define *amalgamation.*

Definition of "Amalgamation"- Given a finite set X, P a partition of X, and ***P*** a partition of P, define amal (P ; ***P***) to be the partition of X whose members consist of the u-function applied to each collection in ***P***. We say that P is "amalgamated" by ***P***.

To demonstrate this operation, let X = {A, B, C, D, E, F, G, H, I, J, K}.

P = {{A, B, C}, {D, E}, {F}, {G, H}, {I, J, K}} and let

P = { {{A, B, C}, {D, E}}, {{F}, {G, H}}, {{I, J, K}} }

Note that ***P*** is literally a partition of P. Thus,

amal (P ; ***P***)

= { u({{A, B, C}, {D, E}}), u({{F}, {G, H}}), u({{I, J, K}}) }

= { {A, B, C} $\cup$ {D, E} , {F} $\cup$ {G, H} , {I, J, K} }

= { {A, B, C , D, E} , {F,G, H} , {I, J, K} }

Note that amal (P ; ***P***) and the original partition P are both partitions of {A, B, C, D, E, F, G, H, I, J, K}. Mathematically, one could identify here that amal (P ; ***P***) is a *coarser* partition than P, meaning that every set in amal (P ; ***P***) is a union of sets in P, and both are partitions of X.

Furthermore, amalgamation serves as a way to coarsen partitions similar to how filter did. Though a different type of function, it also lowers the level of information in a partition by unifying sub-collections of P, and thus lowers the granularity of the partition. Later, when collections of objects are associated to partitions, it will be useful to amalgamate representations of objects so we no longer discern between the objects according to particular attributes. These ideas will come in Chapter 6.

Thus, we have two ways to lower the detail of a partition: filter and amalgamation. Previously, we showed that insert is a way that one can raise the level of detail, being the inverse of filter. Thus, we shall also discuss what it means to have higher granularity in a manner opposite to amalgamation.

Thus, let us continue by defining an inverse operation for amalgamation, *refinement*.

Definition of "Refinement" – Given a set X and partition P' = $\{P_1 \dots P_k\}$ of X, let $\boldsymbol{Q} = \{Q_1 \dots Q_k\}$ be such that each Q_i is a partition of P_i. Define ref (P' ; $\boldsymbol{Q}$) to be the partition of X given by u($\boldsymbol{Q}$). We say that P' is "refined" by $\boldsymbol{Q}$.

To show the inverse operation clearly performed, let

P' = { {A, B, C , D, E} , {F,G, H} , {I, J, K} } and let $\boldsymbol{Q} = \{Q_1, Q_2, Q_3\}$, where

Q_1 = {{A, B, C}, {D, E}}, Q_2 = { {F} , {G, H}} , Q_3 = {{I, J, K}}. Thus,

ref (P' ; $\boldsymbol{Q}$)

= u($\boldsymbol{Q}$)

= u (Q_1, Q_2, Q_3)

= u ({{A, B, C}, {D, E}}, { {F} , {G, H}} , {{I, J, K}})

$= \{\{A, B, C\}, \{D, E\}\} \cup \{ \{F\}, \{G, H\}\} \cup \{\{I, J, K\}\}$

$= \{\{A, B, C\}, \{D, E\}, \{F\}, \{G, H\}, \{I, J, K\}\}$

Note below how the result of ref (P' ; ***Q***) gave us back our original partition in the amalgamation example. Thus, we see pictorially how amalgamation and refinement are inverses of one another.

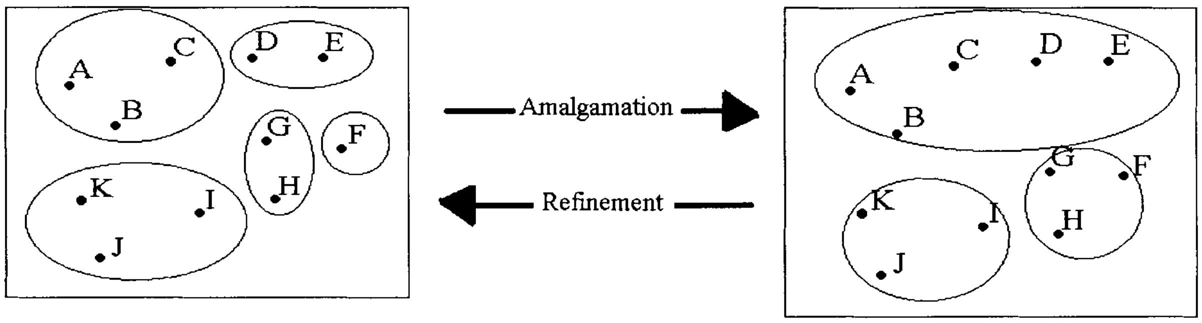


Figure 3.3 Amalgamation unifies sub-collections of a partition, while refinement partitions elements into a new single partition.

Thus, we have established two types of operations that contain a single function and its inverse. Using filter/insert and amal/ref, we shall next construct composition operations on partitions of single sets. These operations are called *simplification/complexification.* Simplification will be our means for developing the partial-order structure and lattice structure of collections of partitions.

3.3 Simplification/Complexification

With the basic operation types filter/insert, and amal/ref, we can now define the more complex operations of interest. Particularly, we will focus on the operation type *simplification/complexification (simp/comp).*

In work done by Stell and Worboys, simplification is used for the purpose of generalizing graphs. As stated as part of their goals, it is important to explore the foundations of the concept "less detailed than", based on the notions of selection (filter) and amalgamation. Using selection (filter) and amalgamation, the two are composed to create simplification (Stell and Worboys 1999).

Thus, we shall also define the composition of filter followed by amalgamation as a *simplification.*

Definition of "Simplification" – Given a set X, and P a partition of X, let F be a subset of P and ***P*** a partition of P\F. Define, simp (P ; F, ***P***) to be equal to the composition amal(filt(P ; F); ***P***). Simp (P ; F, ***P***) is a partition of X\u(F). We say that simp (P ; F, ***P***) is a simplification of P by F and ***P***.

For example, let P = { [0, 2] , [3 , 4] , [5, 6] , [7, 8] , [9, 10]}, F = {[5, 6], [7, 8]}, and ***P*** = { {[0, 2] , [3 , 4]} , {[9, 10]} }. Thus, simp(P;F, $\boldsymbol{P}$)

= amal(filt(P ; F); $\boldsymbol{P}$)

= amal({P\F} ; $\boldsymbol{P}$)

= amal({[0, 2] , [3 , 4] , [9, 10]}; { {[0, 2] , [3 , 4]} , {[9, 10]}})

= { u({[0, 2] , [3 , 4]}) , u({[9, 10]})}

= { [0, 4] , [9, 10]}.

Thus, simp(P;F, $\boldsymbol{P}$) = { [0, 4) , [9, 10]}, and is a partition of X\u(F) = $[0, 4] \cup [9, 10]$.

Stell and Worboys show the application of simplification in graph theory by the filtering and amalgamation of subway train stops in London. The example shows how the composition of these operations generates a more generalized graph. Though manipulation of graphs is beyond the scope of this paper, our work on partitions can eventually allow for similar types of manipulation and could extend to other realms of representation (Stell and Worboys 1999).

Next, note that simplification is only one part of the simp/comp type operation, so we must also define what is meant by the inverse operation, *complexification*.

Definition of "complexification" – Given set X and P' = {P_1 ... P_k}, a partition of X, let $\boldsymbol{Q}$ = {$\boldsymbol{Q}_1$ ... $\boldsymbol{Q}_k$} be such that each $\boldsymbol{Q}_i$ is a partition of P_i . Furthermore, let F be a collection of mutually disjoint sets such that u(F) $\cap$ X = $\varnothing$. Define comp(P' ; $\boldsymbol{Q}$, F) to be equal to the composition ins (ref (P' ; $\boldsymbol{Q}$) ; F). Comp(P' ; $\boldsymbol{Q}$, F) is a partition of X $\cup$ u(F). We say that comp(P' ; $\boldsymbol{Q}$; F) is a "complexification" of P' by $\boldsymbol{Q}$ and F.

Thus, we have defined an operation that should reverse what was performed by simplification. For example, let P' = { [0, 4] , [9, 10]} , let $\boldsymbol{Q}$ = {$\boldsymbol{Q}_1$, $\boldsymbol{Q}_2$ }= {{ [0, 2], [3, 4]}, {[9, 10]} } and let F = {[5 , 6] , [7, 8]}. Thus,

comp(P' ; $\boldsymbol{Q}$; F)

= ins (ref (P' ; $\boldsymbol{Q}$) ; F)

= ins ({[0, 2] , [3 , 4] , [9, 10]}; F)

= {[0, 2] , [3 , 4] , [9, 10]} $\cup$ {[5, 6] , [7, 8]}

= { [0, 2] , [3, 4] , [5, 6] , [7, 8] , [9, 10]}

Thus, comp(P' ; $\boldsymbol{Q}$; F) = { [0, 2] , [3, 4] , [5, 6] , [7, 8] , [9, 10]} is a partition of X $\cup$ u(F) = [0, 10].

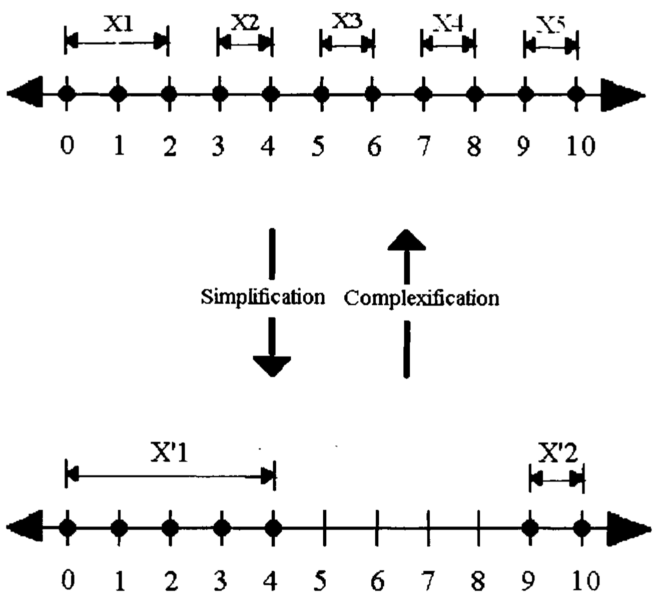


Figure 3.4 *Simplification and complexification are inverse operations of one another, where simplification is filter followed by amalgamation, and complexification is refinement followed by insert.*

Note that complexification is a refinement followed by insert. These are the inverse operations of amalgamation and filter respectively, and complexification is thus the inverse of simplification. This inverse relationship is also evident in the example used, where simplification yielded P' = {[0, 4] , [9, 10]}, and then complexification gave us back our original partition P.

Summary 3.4

In this section, we have looked at partitions and some operations on partitions. Filter/insert is an operation type that removes or adds sets to a partition. Amal/ref unifies sub-collections of a partition or splits apart existing partition sets. Both of these operation types are composed to create the simp/comp operations, where filter followed by amalgamation yields simplification, and refinement composed with insert leads to complexification.

Note also that one of the key differences between this work and work done by Stell and Worboys is the discussion of insert, refinement, and complexification. These operations serve as inverses of filter, amalgamation, and simplification respectively.

In the next section, we will focus specifically on simplification, and prove a number of important properties. We will show that the simp operation not only can be composed to generate a single simplification, but also results in a partial-order structure and lattice structure on partitions.

Chapter 4

ORDER RELATIONSHIPS BASED ON SIMPLIFICATION

4.1 Basic Lemmas

The purpose of this chapter is to look specifically at simp/comp type relationships between partitions, and how a partial-order relation exists in regard to simplification. To prove that we have a partial order structure, we first need to show the fact that for every composition of two simplifications on an arbitrary partition P, there exists a single simplification of P that yields the same result. We will need to prove four necessary lemmas to verify this simplification composition theorem.

First, we will prove that the composition of two filterings of P is the same as a single filtering of P.

Lemma 4.1.1: Let X be a set and P be a partition of X. Assume $F \subseteq P$ and $F' \subseteq P \backslash F$, then

$$\text{filt}(\text{filt}(P ; F) ; F') = \text{filt}(P ; F \cup F')$$

Proof: First, note by the definition of "filter" that filt (P, F) = P \ F. Thus, we can say that $\text{filt}(\text{filt}(P, F), F') = \text{filt}(P\backslash F; F') = (P \backslash F) \backslash F' = P \backslash (F \cup F')$. ♦

Now that we have established that a single filtering can represent the composition of two, we must also show the same for amalgamation. To do so, we must first define the notion of a *sub-partition.*

Definition of "sub-partition" - Given X with partition P, we say $U \subseteq X$ is sub-partitioned by P if some subset of P partitions U.

For instance, P = { [0, 2] , [3 , 4] , [5, 6] , [7, 8] , [9, 10]} partitions the set X = [0 , 10], but it also sub-partitions subsets of X such as [0, 2], [0, 4], and [5, 10], as well as others. This is because sub-collections of P partition these sets. Next, we shall see that each element of an amalgamated partition is sub-partitioned by the initial partition.

Lemma 4.1.2: Given X partitioned by P and Q, if Q is an amalgamation of P and $Q_i \in Q$, then Q_i is sub-partitioned by P.

Proof: If Q = amal(P; $\boldsymbol{P}$), then $\boldsymbol{P}$ is a finite partition of P. Therefore, $\boldsymbol{P} = \{ \boldsymbol{P}_1, \ldots, \boldsymbol{P}_m \}$ and $Q = \{u(\boldsymbol{P}_1), \ldots, u(\boldsymbol{P}_m)\}$. Thus, each Q_i equals some $u(P_i)$, $1 \leq i \leq m$. Furthermore, since each $\boldsymbol{P}_i \subseteq P$, each Q_i is partitioned by $\boldsymbol{P}_i$, and is sub-partitioned by P. ♦

Using this notion of sub-partitions, we can also show that the composition of two amalgamations is the same as some single amalgamation.

Thus Lemma 4.1.3 states the following:

Lemma 4.1.3: Let X be a set and P be a partition of X. If Q = amal (P; ***P***) and R = amal (Q ;***Q***), where ***P*** and ***Q*** are partitions of P and Q respectively, then there exists a partition ***P'*** of P such that R = amal (P; P'). Thus,

$$R = \text{amal}(\text{amal}(P; \boldsymbol{P}) ; \boldsymbol{Q}) = \text{amal}(P, \boldsymbol{P'})$$

Proof: First, since Q = amal (P; ***P***), every element in Q is sub-partitioned by P according to Lemma 4.1.2. Similarly, since R = amal (Q ;***Q***), Q sub-partitions every element of R. Clearly then, P sub-partitions each element of R, since elements in R are unions of elements from Q which are each sub-partitioned by P. Thus, $R = \{u(\varphi_1), \ldots, u(\varphi_m)\}$, where each φ_j is a collection of elements from P that partitions a particular element in R , $1 \leq j \leq m$.

Also, since P and R are partitions, we know that $\varphi_1, \ldots, \varphi_m$ are mutually disjoint. Thus, it is clear that ***P'*** = $\{\varphi_1, \ldots, \varphi_m\}$ is a partition of P such that R = amal(P; ***P'***), and we are done. ♦

Thus, we also see that the composition of any two amalgamations can be expressed as some single amalgamation. Next, our final lemma will show that for every amalgamation followed by filtration, there exists some filtration followed by amalgamation that can generate the same result.

Lemma 4.1.4: Let X be a set and P be a partition of X. Furthermore assume that $\boldsymbol{P}$ is a partition of P and F is a subset of amal(P, $\boldsymbol{P}$). Then there exists F' $\subseteq$ P and a partition $\boldsymbol{P}'$ of P\F' such that

amal (filt (P; F'); $\boldsymbol{P}'$) = filt (amal (P, $\boldsymbol{P}$) ; F)

Proof: First, let $\boldsymbol{P} = \{\Gamma_1, \dots \Gamma_L\}$, where each Γ_i is a collection of elements from P. Thus, Q = amal (P ; $\boldsymbol{P}$) = $\{u(\Gamma_1), \dots, u(\Gamma_L)\}$. Next, without loss of generality let

F = $\{u(\Gamma_1), \dots, u(\Gamma_j)\}$, where $1 \le j \le L$. Let

R = filt (Q ; F) = $\{u(\Gamma_1), \dots, u(\Gamma_L)\} \setminus \{u(\Gamma_1), \dots, u(\Gamma_j)\} = \{u(\Gamma_{j+1}), \dots, u(\Gamma_L)\}$.

Next let F' = $u(\{\Gamma_1, \dots, \Gamma_j\})$ and let $\boldsymbol{P}' = \{\Gamma_{j+1}, \dots, \Gamma_L\}$. Thus we can conclude that

amal (filt (P; F') ;$\boldsymbol{P}'$)

= amal (filt (P; $u(\{\Gamma_1, \dots, \Gamma_j\})$); $\{\Gamma_{j+1}, \dots, \Gamma_L\}$)

= amal ($u(\{\Gamma_{j+1}, \dots, \Gamma_L\})$; $\{\Gamma_{j+1}, \dots, \Gamma_L\}$)

= $\{u(\Gamma_{j+1}), \dots u(\Gamma_L)\}$

= R.

So it follows that R = amal (filt (P; F'); $\boldsymbol{P}'$) = filt (amal (P, $\boldsymbol{P}$) ; F). ♦

4.2 Composition Theorem

Thus, we have proven important lemmas that will help us to easily show that the composition of any two simplifications on a partition P can be expressed as a single simplification of P. Below, we shall show this as our first formal theorem.

Theorem 4.1: First, let Q = simp(P; F ,***P***) and let R = simp(Q; F' ,***Q***). There exists a G and ***R*** such that

simp(simp (P ; F, ***P***);F , ***Q***) = simp(P ; G, ***R***)

Proof: First, note that we need not expound upon the characteristics of ***P***, ***Q***,, F, and F', since we only need our three lemmas to show the proof. Thus, we can first say that

simp(simp (P ; F, ***P***); F , ***Q***)

= amal(filt(amal(filt(P; F); ***P***);F') ***Q***)

= amal(amal(filt(filt(P; F); F''); ***P'***) ***Q***) for some P' and F''.

This is true because of Lemma 4.1.3, where every amalgamation followed by filter can be expressed as some filter followed by amalgamation. Next, we can also say by Lemma 4.1.1 that

simp(simp (P ; F, ***P***); F , ***Q***)

= amal(amal(filt(filt(P; F); F''); ***P'***) ***Q***)

= amal(amal(filt(P; F $\cup$ F''); ***P'***) ***Q***)

Let G = F $\cup$ F''. Finally, by Lemma 4.1.2, we can say that the composition of two amalgamations is the same as some single amalgamation. So there exists ***R*** such that, amal(amal(filt(P; G); ***P'***) ***Q***)

= amal(filt(P; G); ***R***)

= simp(P; G, ***R***). ♦

Thus, for the composition of two simplifications, there exists a single simplification that accomplishes the same result. This fact is important for many reasons. The partial-order proof soon to follow depends on this fact. Also, queries related to simplifications of object representations can rely on the notion that the composition of a finite number of simplifications could be expressed as a single simplification. This result easily follows by induction.

As an example of the theorem, let's look at our favorite example, a partition

P = { [0, 2] , [3 , 4] , [5, 6] , [7, 8] , [9, 10]} on the set X = [0,10].

Let

F = {∅}

P = { {[0, 2] , [3 , 4]} , {[5, 6] , [7, 8]} , {[9, 10]} }

F' = {[5, 8]}

P' = {{[0, 4]} , {[9, 10]} }

Thus, we can say that

simp(simp (P ; F, ***P***);F , ***P'***)

$= \text{simp}(\{[0, 4], [5, 8], [9, 10]\}; F, \boldsymbol{P}')$

$= \{[0, 4], [9, 10]\}$.

Our result is a partition of the set $X = [0, 4] \cup [9, 10]$. From Theorem 4.1, we know that there exist G and $\boldsymbol{R}$ such that $\text{simp}(P; G, \boldsymbol{R}) = \{[0, 4], [9, 10]\}$. Letting $G = \{[5, 6], [6, 7]\}$, and $\boldsymbol{R} = \{\{[0, 2], [3, 4]\}, \{[9, 10]\}\}$,

$\text{simp}(P; G, \boldsymbol{R})$

$= \text{amal}(\{[0, 2], [3, 4], [9, 10]\}; \boldsymbol{R})$

$= \{[0, 4], [9, 10]\}$

Thus, we were able to simplify P in one step instead of two. On the following page, you can see this example pictorially and how the simplification composition can be accomplished in one step.

We have verified Theorem 4.1, and now we can use this notion to prove that the simplification relation is a partial order.

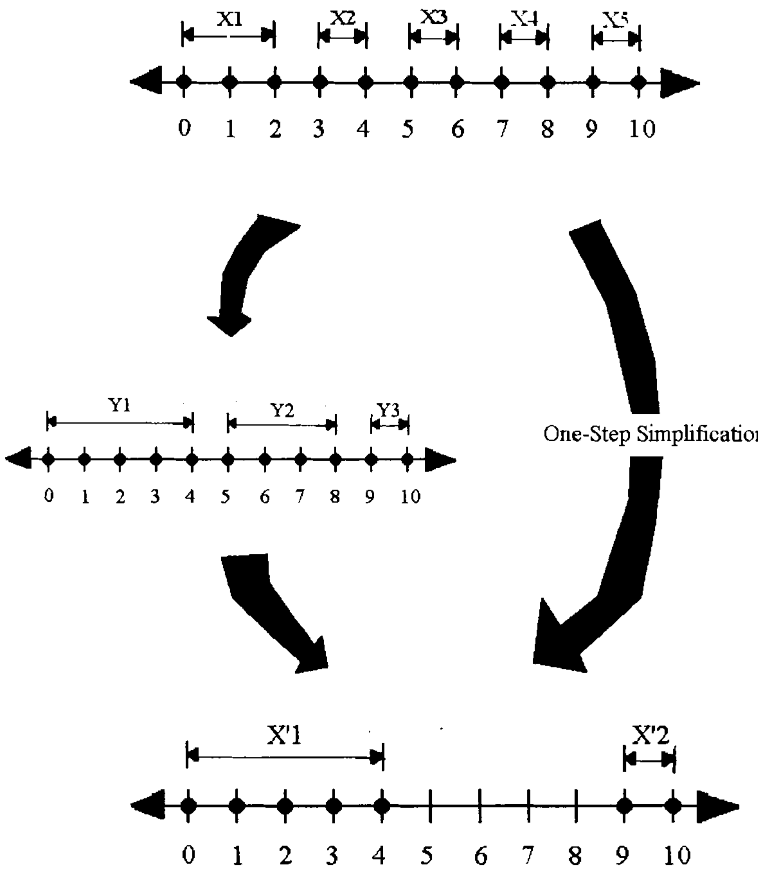


Figure 4.1 *The composition of two or more simplifications can be accomplished in one step with a single simplification.*

4.3 Partial-Order Structure

Thus far, we have mentioned the notion of order relationships but have not considered what exactly shall define one partition to be less than another. In terms of simplification, we shall consider the binary relation $\leq_S$ to mean that

P' $\leq_S$ P if and only if P' = simp(P;F, *P*).

For example,

$\{[0, 4], [9, 10]\} \leq_s \{[0, 2], [3, 4], [5, 6], [7, 8], [9, 10]\}$ because

$\{[0, 4], [9, 10]\} = \mathrm{simp}(P;F, \boldsymbol{P})$

where $F = \{[5, 6], [7, 8]\}$ and $\boldsymbol{P} = \{ \{[0, 2], [3, 4]\}, \{[9, 10]\} \}$.

Thus, we have a way of comparing the relationship of partitions via simplification, and therefore one partition can be considered less than another partition. In fact, we can show that this order relation is a partial-order. In mathematics, a *partial order* $\leq$ on a set X is a binary relation that is reflexive, anti-symmetric, and transitive, i.e., it holds for all a, b and c in X that:

$a \leq a$ (reflexivity)

if $a \leq b$ and $b \leq a$, then $a = b$ (antisymmetry)

if $a \leq b$ and $b \leq c$, then $a \leq c$ (transitivity)

A set with a partial order on it is called a partially ordered set, poset, or, often, simply an ordered set (Birkhoff 1967).

Thus, our goal is to show that $\leq_s$ creates a partial order structure on the set of partitions of a domain D and its subsets. Let's call this collection $\boldsymbol{U}$. To prove that the partial order properties hold for $\boldsymbol{U}$, we shall look at each one and show they are true according to how we have defined $\leq_s$. We will call this Theorem 4.2.

Theorem 4.2: If D is a finite set, then $\boldsymbol{U}$, the collection of all partitions of D and every $X \subseteq D$, is a partially-ordered set according to the $\leq_S$ relation.

Proof:

1) Reflexive Property

First, let P be a partition of $X \subseteq D$. Note that if $F = \varnothing$ and $\boldsymbol{P} = \{P\}$, then simp(P; F ; $\boldsymbol{P}$) = amal(filt(P; $\varnothing$) ; {P}) = P. Thus, by our order definition, $P \leq_S P$, and therefore the reflexive property holds.

2) Anti-symmetric Property

First, it is given that P' = simp(P; F , $\boldsymbol{P}$) and P = simp(P'; F', $\boldsymbol{P'}$). Note if we substitute P into P', then P' = simp(simp(P'; F', $\boldsymbol{P'}$) ; F , $\boldsymbol{P}$). Furthermore, by Lemma 4.1.1 and Theorem 4.1, P' = amal (filt (P'; $F \cup F'$);$\boldsymbol{R}$). Thus, $F \cup F' = \varnothing$, which implies $F = \varnothing$ and $F' = \varnothing$.

Next we can say that since P' = simp(P; $\varnothing$, $\boldsymbol{P}$), then P' = amal(P; $\boldsymbol{P}$). By Lemma 4.1.2, we know that every $P'_i \in P'$ is sub-partitioned by P. Thus, $P'_i = u(\boldsymbol{P}_i)$, where each $\boldsymbol{P}_i \subseteq P$, $1 \leq i \leq n$. Also, since P = simp(P'; $\varnothing$, $\boldsymbol{P'}$), then P = amal(P';$\boldsymbol{P'}$). This implies that every $p \in \boldsymbol{P}_i$ is sub-partitioned by P'. P'_i covers p though, so $P'_i = p$, and each subset $\boldsymbol{P}_i$ is a single element P_i in P. Thus, each $P'_i = P_i$, and it follows that P = P'.

3) Transitive Property

First, it is given that P' $\leq_S$ P and , P'' $\leq_S$ P'. If this is true, then P' = simp(P; F , ***P***) and P'' = simp(P'; F' , ***P'***). Thus, P'' = simp(P'; F' , ***P'***) = simp(simp(P; F , ***P***); F' , ***P'***). By Theorem 4.1 we know that the composition of two simplifications can be expressed as a single simplification. Thus, there exist G and ***R*** such that

P'' = simp(simp(P; F , ***P***); F' , ***P'***) = simp(P; G , ***R***), and by definition, P'' $\leq_s$ P.♦

Thus, we have shown that ***U*** has a partial-order structure via simplification. When we compare two or more partitions, we can compare their order relationship according to simplification and know that reflexive, anti-symmetric, and transitive properties hold. Note though that two partitions may also be completely unrelated in regard to simplification ordering. For instance, the partitions

P = {[0, 2] , [3 , 4] , [5, 6] , [7, 8] , [9, 10]} and

P'= {[0, 1] , [2, 3] , [4, 8] , [9, 10]}

are not order-related in terms of simplification because neither one can be simplified to generate the other. Thus, some partitions are simplification order-related, and some are not. This is what is meant by a "partial" order.

Though partitions P and P' are not order related, they are still related in terms of upper and lower bounds. We shall see in the next section that every pair of partitions has a greatest lower bound and a least upper bound according to simplification partial-order.

4.4 Summary

In this section, we have shown that for the composition of two simplifications, there exists a single simplification providing the same result. To do so, we developed four key lemmas: composition of two filtrations, a sub-partitioning lemma, composition of two amalgamations, and the interchanging of amalgamation followed by filter with filter followed by amalgamation. Finally, we used the simplification composition theorem to prove that $\leq_s$ imposes a partial-order structure on $\boldsymbol{U}$, the collection of all partitions of a finite domain D and its subsets.

We will use the notion of partial-order in the next section when we also show that $\boldsymbol{U}$ has a lattice structure and is complete. In other words, we will see that every collection of partitions in $\boldsymbol{U}$ has a greatest lower bound and a least upper bound. To do so will involve the creation of some new definitions, as well as proving a number of initial lemmas. Once we see that a collection of partitions is a lattice and $\boldsymbol{U}$ is complete, we then will have a strong mathematical framework on partitions which can then be applied to representations of objects.

Chapter 5

SIMPLIFCATION LATTICES OF PARTITIONS

5.1 Background

When talking about the ordering of partitions in regard to simplification, it is not only useful to show that partitions would form a partially-ordered set, but also to look at the existence of upper and lower bounds. Such information can help categorize partitions by showing which ones share common simplifications or complexifications.

More specifically, we desire to see if every pair of partitions has a least upper bound and a greatest lower bound. Using this fact, along with the partial-order structure, verifies the existence of a lattice structure. A *lattice* is defined as a partially-ordered where every pair of elements has a greatest lower bound (g.l.b.) and least upper bound (l.u.b.).

Not only can we prove the existence of a lattice structure, but we can also show that the structure is complete. A partially-ordered set in which every subset that is bounded from above has a least upper bound is called *complete*. Note it is true that every subset of a complete set that is bounded below also has a greatest lower bound. It is also true that if a finite partially ordered set has a maximum and minimum element, and every pair of elements has a g.l.b., then the set is complete (Birkhoff 1967).

In this paper, we have already shown the set ***U*** with $\leq_s$ is a partially ordered set. Recall that ***U*** is the collection of partitions of a domain D and all of its subsets. Since ***U*** is finite, the most refined partition in ***U*** exists as the partition of D made up of all single point subsets. Likewise, the empty set would be the coarsest partition. Thus, ***U*** has a maximum and minimum element. Therefore, our goal in this chapter is to show that every pair of elements has a g.l.b., and this will show that ***U*** is complete, and thus inherently a lattice.

The reason behind this interest has much to do with the application of this material, and how representations of objects can be actualized and labeled according to their attributes. If we can determine which representations are more simplified or complexified than other representations, we can be mathematically certain there exist upper and lower bounds for pairs of representations, which could become important for simplification-related queries in future work.

5.2 Key Lemmas

To show that a finite collection of partitions forms a lattice structure, we first must develop a number of definitions and prove some key lemmas. Our goal is to create and prove important facts that will increase our understanding of simplification-related properties of partitions. Furthermore, these properties relate partitions to one another, and help develop the lattice proof piece by piece.

First, recall the term "sub-partition" from Chapter 4. Given a finite set X with partition P, we say $S \subseteq X$ is sub-partitioned by P if some subset of P partitions S. We shall expand our understanding of this definition in the lemma below.

Lemma 5.1.1 If U_1 and U_2 are sub-partitioned by P, then so are $U_1 \cup U_2$, $U_1 \cap U_2$, $U_1 \setminus U_2$, and $U_2 \setminus U_1$.

Proof: Given U_1 and U_2 are sub-partitioned by P, by definition, U_1 is partitioned by $C_1 \subseteq P$ and U_2 by $C_2 \subseteq P$. Thus, C_1 and C_2 both contain mutually disjoint sets, and it is clear that $C_1 \cup C_2$ and $C_1 \cap C_2$ both contain mutually disjoint sets too (both being subsets of P). Furthermore, $u(C_1) = U_1$ and $u(C_2) = U_2$, so $u(C_1 \cup C_2) = U_1 \cup U_2$, and $u(C_1 \cap C_2) = U_1 \cap U_2$. Thus $C_1 \cup C_2$ partitions $U_1 \cup U_2$ and $C_1 \cap C_2$ partitions $U_1 \cap U_2$. By definition, $U_1 \cup U_2$ and $U_1 \cap U_2$ are both sub-partitioned by P.

Also, since C_1 partitions U_1, and C_2 partitions U_2, and both C_1 and C_2 are subsets of P, it follows that $C_1 \setminus C_2$ is a collection of mutually-disjoint sets such that $u(C_1 \setminus C_2) = U_1 \setminus U_2$. Thus, $U_1 \setminus U_2$ is sub-partitioned by P, and by symmetry, $U_2 \setminus U_1$ is sub-partitioned as well. ♦

Thus, we see that the union, intersection, and difference of two sub-partitioned sets are also sub-partitioned sets. This notion can be easily generalized by induction to show that the union of finitely many sub-partitioned sets is sub-partitioned, as well as the intersection of finitely many sets.

Next, we are not only interested in general sub-partitioning lemmas, but also in sets that are "minimally" sub-partitioned, meaning the smallest possible sets sub-partitioned by two different partitions. We shall call this a *min(P, P')* subset.

Definition of "min(P, P') subset"- Given partitions P,P' of X, if $S \subseteq X$ is sub-partitioned by P and P', but no proper nonempty subset of S is sub-partitioned by P and P', then S is called a min(P, P') subset.

Defining such a set is important for our development of the lattice proof. With this notion, we will eventually show that greatest lower bounds exist for two partitions of the same set. Thus, min(P, P') sets will play a pivotal role in the construction of the greatest lower bound.

Next we shall look at a lemma related to min(P,P') subsets:

Lemma 5.1.2: Given partitions P, P' of X and $P_j \in P$, the intersection of all of the sets sub-partitioned by P and P' that contain P_j is a min (P, P') subset containing P_j as a subset.

Proof: Note that X itself is sub-partitioned by both P and P', and $P_j \in P$ implies that $P_j \subseteq X$. Therefore, X itself is one of the sets making up the intersection, and the intersection exists. Also, we know that if two sets are sub-partitioned by P and P', then their intersection is sub-partitioned by P and P' too (Lemma 5.1.1). Thus, by induction it is clear that the intersection of finitely many sets sub-partitioned by P and P' would also be sub-partitioned by P and P'. Therefore, the intersection of all the subsets of X that contain P_j and are sub-partitioned by P and P' is a set that is sub-partitioned by P and P'. Thus, we have a set containing P_j that is sub-partitioned by P and P' that is the intersection of all such sets. Call it I.

Claim: I is a min(P, P') set. Assume not. Thus there exists $J \subset I$ where J is a min(P, P') set. Clearly, either $P_j \subseteq J$ or $P_j \cap J = \varnothing$, since otherwise it would imply that P does not subpartition J. If $P_j \subseteq J$, that contradicts I being the intersection of all min(P, P') sets containing P_j, since $J \cap I = J$ and J is a proper subset of I. If $P_j \cap J = \varnothing$, then by Lemma 5.1.1., $I \setminus J$ is sub-partitioned by P and P', it contains P_j , and it is contained in I. This again contradicts I being the intersection of all min(P, P') sets containing P_j, since $I \cap (I \setminus J) = I \setminus J$.

Thus, we can conclude that I is a min(P, P') set. ♦

The graphic below (Figure 5.1) demonstrates an example of this lemma. Note all of the sub-partitioned subsets that contain P_j, and I, the intersection of these sets.

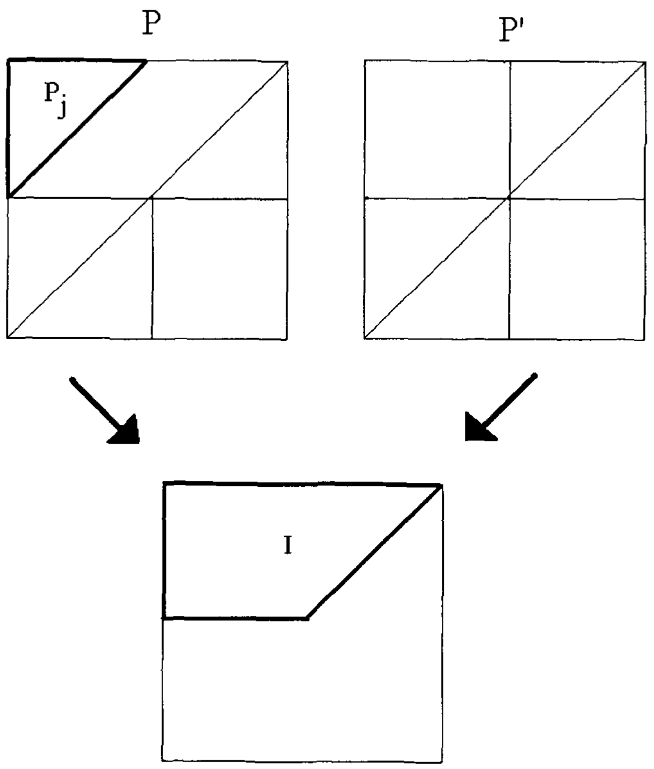


Figure 5.1 *Given that P and P' partition the same set, note that $P_j \in P$ belongs to many subsets sub-partitioned by both P and P', but I is the intersection of all such sets.*

In fact, a partition that has all min(P, P') elements has notable properties in relation to partitions P and P'. Lemma 5.1.3 below shows how a collection of all min(P, P') sets is related to simplification ordering.

Lemma 5.1.3: Given P and P' partitions of X, let $\mu(P, P')$ be the collection of every possible min(P,P') subset. $\mu(P, P')$ is a partition of X, and $\mu(P, P') \leq_s P$, $\mu(P, P') \leq_s P'$.

Proof: First, note that $\mu(P, P')$ is non-empty by Lemma 5.1.2, since X is sub-partitioned by both P and P'. Thus, we must also show that $\mu(P, P')$ is a partition of X. Since $\mu(P, P')$ consists of every possible min(P, P') set, every element of P is covered by some element of $\mu(P, P')$. Thus, since each element of $\mu(P, P')$ is a subset of X, $u(\mu(P, P')) = X$.

Next, note that since every element of $\mu(P, P')$ is sub-partitioned by P and P', then the intersection of any two elements is also sub-partitioned by P and P' (Lemma 5.1.1). Since $\mu(P, P')$ has distinct elements, the intersection of any two of them must be either a proper subset of the elements or empty. If this intersection were non-empty though, this would imply that there exists a subset of both elements that P and P' would sub-partition, which would contradict the definition of a min(P, P') set. Thus the sets in $\mu(P, P')$ are mutually disjoint, and $\mu(P, P')$ is a partition of X.

Finally, since elements in $\mu(P, P')$ are sub-partitioned by P and P', each element in $\mu(P, P')$ is a union of elements from both P and P'. Thus, $\mu(P, P') = \{u(P_1),\ldots, u(P_n)\} = \{u(P'_1),\ldots, u(P'_n)\}$, where each $P_i \subseteq P$ and each $P'_i \subseteq P'$, $1 \leq i \leq n$. Thus, $\boldsymbol{P} = \{P_1,\ldots, P_n\}$ partitions P and $\boldsymbol{P'} = \{P'_1,\ldots, P'_n\}$ partitions P'. Finally, by definition, we can say that $\mu(P, P') = \text{amal}(P, \boldsymbol{P}) = \text{amal}(P', \boldsymbol{P'})$. Thus, $\mu(P, P') = \text{simp}(P,\varnothing, \boldsymbol{P}) = \text{simp}(P',\varnothing, \boldsymbol{P'})$ and we can say $\mu(P, P') \leq_s P$ and $\mu(P, P') \leq_s P'$. ♦

So $\mu(P, P')$ is an amalgamation of P and an amalgamation of P'. This step is important, but we can say something even stronger about $\mu(P, P')$. We can show it is not only a lower bound of both P and P' on X, but it is also a greatest lower bound. Note Lemma 5.1.4 below.

Lemma 5.1.4: If $Q \leq_s P$, $Q \leq_s P'$, and $Q \neq \mu(P, P')$, then $Q <_s \mu(P, P')$.

Proof by Contradiction: Assume $\mu(P, P') <_s Q$. Thus, every element of $\mu(P, P')$ is sub-partitioned by Q by Lemma 4.1.2. Furthermore, since the inequality is strictly "less than", there exists a non-trivial sub-partitioning of some element $\alpha \in \mu(P, P')$ by Q. Therefore, there exists $q \in Q$ such that $q \subset \alpha$. Since, $Q \leq_s P$ and $Q \leq_s P'$, q is sub-partitioned by both P and P'. This implies that q is a proper subset of α, which is sub-partitioned by both P and P' according to Lemma 4.1.2 and Lemma 5.1.3. This contradicts that given fact that α is a min(P, P') subset, and therefore $Q <_s \mu(P, P')$. ♦

The previous lemmas proved important facts about lower bounds of two partitions of the same set. Two partitions may not necessarily cover the same underlying sets though. The following lemma will consider arbitrary partitions of separate sets.

Lemma 5.1.5 Let P be a partition of X and P' be a partition of X'. If simp(P; F , ***P***) = simp(P'; F' , ***P'***), then $u(P \backslash F) = u(P' \backslash F') \subseteq X \cap X'$, and $u(P \backslash F)$ is sub-partitioned by P and P'.

Proof: Note that if simp(P; F , ***P***) = simp(P'; F', ***P***'), then amal (P\ F , ***P***) = amal(P'\ F',***P'***). Since amalgamation does not change the underlying set, u(P\F) = u(P'\F'). Furthermore, since u(P\F) $\subseteq$ X and u(P\F) $\subseteq$ X', u(P\F) $\subseteq$ X $\cap$ X'. Finally, note that u(P\F) is partitioned by P\F and P'\F', so it is obviously sub-partitioned by P and P'. ♦

5.3 Simplification Lattice Proof

With all of the lemmas from the last section, we now have a strong foundation of proof for the lattice theorem. Lemma 5.1.4 is particularly important, since it guarantees that μ(P, P') is a greatest lower bound when P and P' both partition the same set. Since the lattice proof deals with arbitrary partitions though, we will have to assume that P and P' are partitions of different sets.

In the theorem below, we will show that ***U*** with $\leq_s$ is a lattice. We already know that ***U*** with $\leq_s$ is a partially-ordered set, so we need only show that every pair of elements has a greatest lower bound (g.l.b.) and a least upper bound (l.u.b.). At the same time, we will also show that ***U*** is complete, where any collection in ***U*** has a g.l.b and l.u.b (Birkhoff 1967).

Theorem 5.1: Let ***U*** be the collection of partitions on a finite set D and all of its subsets. ***U*** has a lattice structure and is complete in regard to the $\leq_s$ relation.

Proof: Given P partitions $X \subseteq D$ and P' partitions $X' \subseteq D$, let M be the union of all sets subpartitioned by both P and P'. We shall call this type of set a "maximal" subpartitioned set. Note that M is also sub-partitioned by P and P', since it is a finite union of such sets (Lemma 5.1.1). Thus, there exist $P_k \subseteq P$ and $P'_k \subseteq P'$ that partition M. Furthermore, there exist $F_k \subseteq P$ and $F'_k \subseteq P'$ such that $P \setminus F_k = P_k$ and $P' \setminus F'_k = P'_k$.

Next, let LB(A,B) represent the set of all lower bounds of two partitions A and B. Claim: $LB(P, P') = LB(P_k, P'_k)$.

First, if arbitrary $Q \in LB(P_k, P'_k)$, then the fact that $P_k \leq_s P$ and $P'_k \leq_s P'$ implies that $Q \leq_s P$ and $Q \leq_s P'$ by transitivity. Thus, $LB(P_k, P'_k) \subseteq LB(P, P')$. Next, let Q' be an arbitrary partition less than both P and P'. Thus $Q' = \text{amal}(P \setminus F ; \boldsymbol{P}) = \text{amal}(P' \setminus F' ; \boldsymbol{P'})$. Since M is the largest sub-partitioned subset of $X \cap X'$, by Lemma 5.1.5, $P \setminus F \subseteq P \setminus F_k$. This implies that $F_k \subseteq F$, which tells us that there exists F_m such that $F_m \cap F_k = \varnothing$ and $F_k \cup F_m = F$. Thus,

$$Q = \text{amal}(P \setminus F ; \boldsymbol{P}) = \text{amal}(\text{filt}(P ; F ; \boldsymbol{P}) = \text{amal}(\text{filt}(P ; F_k \cup F_m); \boldsymbol{P})$$
$$= \text{amal}(\text{filt}(\text{filt}(P ; F_k) ; F_m); \boldsymbol{P}) = \text{amal}(\text{filt}(P \setminus F_k ; F_m); \boldsymbol{P}) = \text{amal}(\text{filt}(P_k ; F_m)); \boldsymbol{P})$$
$$= \text{simp}(P_k ; F_m , \boldsymbol{P}).$$

So, $Q \leq_s P_k$, and by similar methods, $Q \leq_s P'_k$. Thus, $LB(P, P') \subseteq LB(P_k, P'_k)$, and we can conclude that $LB(P, P') = LB(P_k, P'_k)$. By Lemma, 5.1.4, we know that P_k and P'_k have $\mu(P_k, P'_k)$ as a greatest lower bound. Thus, since P and P' share the same lower bounds, any two arbitrary partitions in $\boldsymbol{U}$ have a greatest lower bound.

Since we have shown that every set of two elements has a g.l.b., if we assume that every set of n-1 elements has a g.l.b., we can show that every set of n elements does as well. Thus, every finite subset of ***U*** has a greatest lower bound by induction.

Furthermore, since ***U*** with $\leq_s$ is finite, it has the greatest lower bound property, meaning every bounded subset has a greatest lower bound. Also, it is a common result that any partially ordered set with the g.l.b property also has the least upper bound property. Thus, ***U*** is complete since not only every pair of elements has a g.l.b. and l.u.b., but also every sub-collection of ***U*** (Birkhoff, 1967).♦

We have shown that there exists greatest lower bound a least upper bound for any subset in ***U***. For example, if X = {a, b, c}, then every possible partition of X and its subsets forms a lattice.

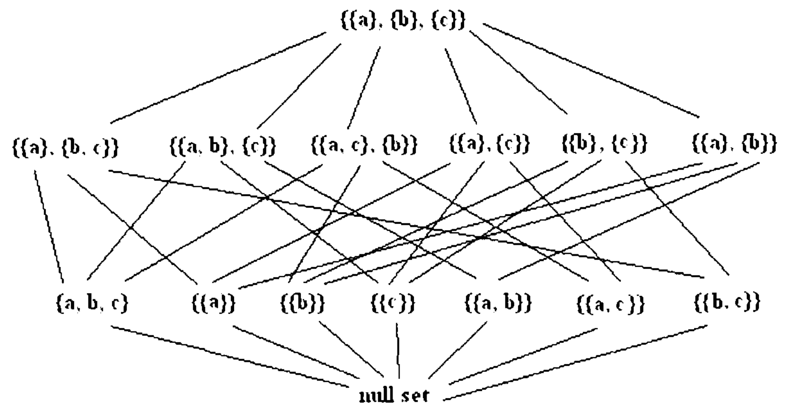


Figure 5.2 The simplification lattice of the three point set {a, b, c}.

Note how {{a}, {b}, {c}} is the most "complex" partition since it is a collection of all the single point sets, and note that the null set as the most "simplified". Also, notice that every subset has a greatest lower bound and a least upper bound.

For instance, {{a, b, c}} and {{a}} have {{a}, {b, c}} as a least upper bound and the null set as a greatest lower bound. {{a}, {b, c}}, {{a, c}, {b}}, and {{a, b}, {c}} have {{a}, {b}, {c}} as a least upper bound, and {{a, b, c}} as a greatest lower bound.

5.4 Summary

In this chapter, we developed important definitions and lemmas that allowed us to prove a very important theorem. If ***U*** is the collection of partitions on a set X and all of its subsets, then ***U*** has a lattice structure in regard to the $\leq_s$ relation and is complete. Through the basic example of partitions of a three point set and its subsets, we saw how such a set behaved as a lattice structure, and noted that every pair of elements had a greatest lower bound and least upper bound.

Next, we shall look at a more applied structure on which to operate. Instead of partitions on basic number sets, we shall now look at views of view domains, and the resulting representations depending upon the coding function. Also, we shall see how simp/comp operations work within this realm, and see how to change the level of detail of a representation.

Chapter 6

VIEWS AND REPRESENTATIONS

6.1 Views

So far, we have developed a solid base of mathematical theory in regard to simplification of partitions. The definitions we have created can also relate to product partitions of finite product spaces. This fact is important since we desire to relate what we have done to queries in geographic information systems associated to finite attribute domains.

We want to be able to study collections of objects according to their attributes, and be able to operate on representations of these objects. We also want to have the freedom to change the level of detail of a representation of objects according to their attributes. Therefore, we must use what we have developed thus far for simplification of partitions not only to select specific types of objects to look at, but also to make desired levels of discernment between them.

First, there must be some way to determine the sets of attributes we are interested in. This space is a subset of the cross product of attribute domains, and is called the *view domain.*

Definition of "View Domain" - A view domain is an finite n-product space

$V_D = X_1 \times X_2 \times \ldots \times X_n$, where each X_i is a subset of the attribute domain A_i, $1 \leq i \leq n$.

Thus, just as a single finite set X was used earlier to denote the underlying set for an arbitrary partition P, each X_i will be partitioned by some P_i. The collection of every n-product composed of subsets from each P_i is called a *view.*

Defintion of "View" – Given a view domain $V_D = X_1 \times X_2 \times \ldots \times X_n$, let $P_1,\ldots,P_n$ be partitions of $X_1,\ldots,X_n$ respectively. Thus, a view V is a product partition of V_D, where each partitioned subset of V is an n-product $\beta_1 \times \ldots \times \beta_n$, where each $\beta_i \in P_i$.

Thus, a view is simply a partition of a view domain, and its properties are no different than the arbitrary partitions we studied earlier. The major difference is in the semantics, since now we are talking about an applicable concept in a GIS.

In fact, since V is a partition of V_D , we can also say that all of the previous lemmas and theorems that held for arbitrary partitions also hold for V. This includes the idea that, given a finite attribute domain, the collection of every view on every view domain has a partial order structure and a complete lattice structure. These facts are very important when trying to find necessary views that contain the desired amount of information about collections of objects according to their attributes.

6.2 Actualization of Views

Now that we have developed all of our work on partitions and views, it is important that we also briefly mention the topic of actualization. The previous work makes sense when studying partitions of basic sets and product spaces, but when we also try to connect the work to real scenarios, the process can be equally as challenging.

First, we must figure out which objects would be seen in a representation, given some specific view. Since a view is a partition of a subset of the attribute domains, we are only interested in objects related to the underlying view domain. Particularly, we want to see to which partitioned subset the objects' codings belong in the view.

The representation of objects whose codings belong to the view is called the *view actualization.*

Definition of "View Actualization" - Given a universe of objects ***O***, n attributes, and attribute domains $A_1, \ldots, A_n$, the view actualization is the labeling of objects whose codings lie within particular partitioned subsets of the view.

Thus, a view labeling function is not a function on the objects, but rather the partitioned subsets of the view.

6.3 Example

Let's imagine that there were 50 important buildings in some city. These buildings have 4 attributes of interest: financial value (in millions), location (numbered regions), zoning, and age in years. The attribute domains for each of the four attributes are,

$A_1 = [1, 20]$,

$A_2 = [1, 16]$,

$A_3 = \{\text{residential, commercial, historical, recreational}\}$, and

$A_4 = [1, 200]$

We begin our query by looking for all of the buildings that are in the view domain: $V_D = [5, 10] \times \{16\} \times \{\text{residential, commercial}\} \times [1, 90]$. Notice that V_D is a subset of the attribute domain $A_1 \times A_2 \times A_3 \times A_4$. Thus, we are looking for buildings between 5 and 10 million dollars in value, located in region 16, with residential or commercial zoning, and between 1 and 90 years old.

Next, we partition each part of V_D. Let $P_1 = \{[5, 6], [7, 8], [9, 10]\}$, $P_2 = \{\{16\}\}$, $P_3 = \{\{\text{residential}\}, \{\text{commercial}\}\}$, and $P_4 = \{[1, 30] , [31, 60], [61, 90]\}$. Thus, a view would consist of all partitioned cross-product subsets that can be generated with each partition. For instance,

$\{[5, 6] \times \{16\} \times \{\text{residential}\} \times [1, 30]\}$,

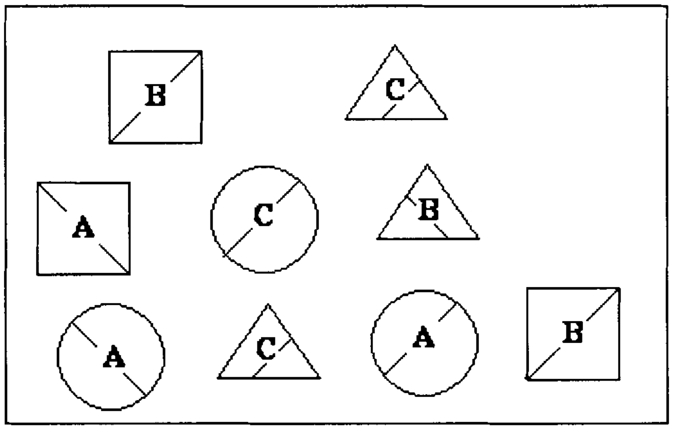
$\{[5, 6] \times \{16\} \times \{\text{commerical}\} \times [61, 90]\}$, and

$\{[9, 10] \times \{16\} \times \{\text{residential}\} \times [31, 60]\}$

would be some of the sets contained in the view.

Thus, there are 18 partitioned subsets of V. Next, to actualize the view, each one of the partitioned subset in P_1, P_2, P_3, and P_4 is assigned a label. This label is applied to any object whose coding is contained in the set.

Thus, let us assume that each partition has a labeling where the value is represented by “letters”, zoning is represented by “sloped markings”, and age is represented by “shape”. Also suppose that there were 9 buildings whose codings were contained in the view domain. Thus, depending on in which partitioned subset each coding was contained, the following representation could result.



A	[5, 6]
B	[7, 8]
C	[9, 10]
pos. slope	{residential}
neg. slope	{commercial}
square	[0, 30]
circle	[31, 60]
triangle	[61, 90]

Figure 6.1 *A representation of objects is based on the view actualization, which depends on which objects have codings contained in various partitioned subsets of the view.*

Note that the system of labeling here is somewhat different than representations in other spatial engineering literature. Amalgamation is usually represented by combining objects into one entity rather than combining labels. This example shows that the labeling function can be defined in a number of ways. Certainly if there were only one attribute of interest, one might combine objects into single entities.

The study of determining which labeling functions are best to use or how to define practical labeling functions is beyond the scope of this work. Our example serves as only a sample of one type of representation. A person could choose to label partitions in a number of ways, and might include labeling based on how objects relate to one another in terms of containment, connectedness, and nearness (Ramalingam 2002).

With that said, we are not only interested in the representation's appearance, but also note that simplification/complexification can be applied to views, and the representation would be changed as a result. For instance, if the previous view were simplified so that partitioned subsets with {commercial} in them were filtered, and [5, 6] and [7, 8] were amalgamated, the result would appear like Figure 6.2 below.

Along with this example, we also know that we can determine other facts as well, such as whether or not we can find greatest lower bounds for collections of representations. Since this is a known fact for views, we can say the same for the seen

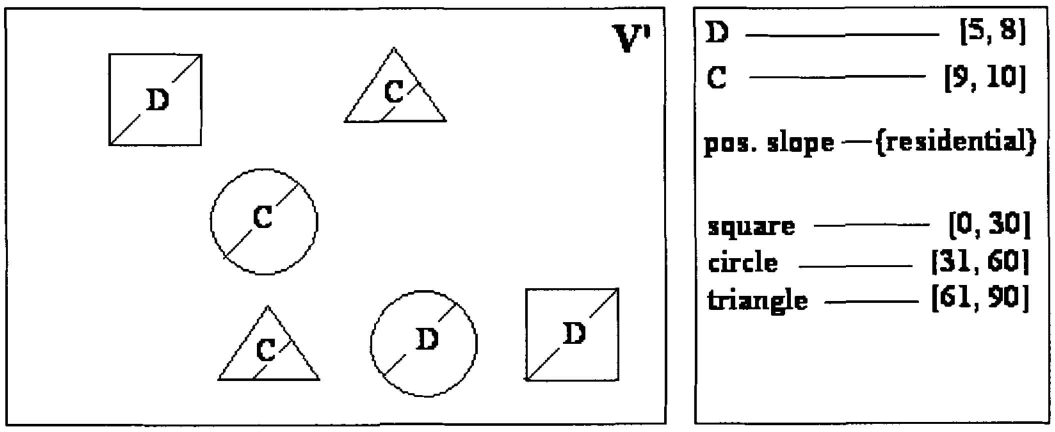


Figure 6.2 *The representation shows the result of a simplification of the view V.*

representation. Thus, a person seeking to find representations that capture a common thread among many representations could seek the view guaranteed by the underlying lattice structure.

Thus, if a person wanted to see the greatest lower bound of $\{V_1, \ldots, V_m\}$, they would look for the view $\mu(V_1, \ldots, V_m)$. The result would be a representation that would serve as greatest lower bound of all the representations of $V_1, \ldots, V_m$.

6.4 Summary

In this section, we have looked at view domains and views. A view domain is simply a subset of the cross product of attribute domains. Likewise, a view is a partition of a view domain. Using this notion, we discussed some general ways one might represent objects based on a given view. If an object's coding is contained in the view domain, then the object's actualization is based on the view. Also, each subset of the view has a particular label, so objects with codings contained in the subset inherit the label.

A geographic example was also used here, where buildings were actualized according to where their codings were contained in the view. Simplification was also applied to the view, and the results were seen in the representation. Thus, representations that can act as greatest lower bounds or least upper bounds of view actualizations exist because they exist in the underlying view structure.

CONCLUSIONS AND FURTHER WORK

The work described in this paper has been concerned with the creation of a mathematical model that formally produces generalized representations of geographic information. In work done by Stell/Worboys and Ramalingam, much of this work has already been modeled, but not in the manner that has been done here. Inverse operations such as insert, refinement, and complexification were defined and studied.

The manipulation of views presents an interesting way to generalize objects and formally prove some of the properties related to the model. In particular, we have shown that simplification generates a partial-order and lattice structure on views and the resulting representation as well. Though there is much more to show in this area, it establishes the existence of greatest lower bounds and least upper bounds, and allows the possibility for future math-related work, including the study of infinite attribute domains and infinite partitions of those domains.

Our approach could be improved. The syntax of the operations becomes quite cumbersome when large collections of sets are either filtered or inserted. Furthermore, the notation required for amalgamating or refining large sets is also tedious. The next step in the work is to consider methods of easily describing the operations on partitions, but also preserve the semantics of the model. Secondly, there needs to be a method of characterizing amalgamations based on the notion of object containment, connectedness, and nearness (Ramalingam 2002), while still preserving the existing theory. Finally, view-labeling functions could be more fully developed in regard to semantic practicality.

Deciding which labeling is best for conveying a representation of a view presents many open-ended challenges.

BIBLIOGRAPHY

G. Birkhoff (1967) *Lattice Theory* . Colloquium Publications, Providence, pp. 112-115.

C. Ramalingam (2002) *Modeling Multiple Granularities of Spatial Objects.* M.S. Thesis, University of Maine, Orono.

J. Stell and M. Worboys (1999) Generalizing graphs using amalgamation and selection. In: R. Guting, D. Papadias, and F. Lochovsky (Eds.), *Advances in Spacial Databases, 6th International Symposium, SSD'99*, Hong Kong, China, pp. 19-32.

BIOGRAPHY OF THE AUTHOR

Gabriel Perrow was born in Millinocket, Maine. He and his wife Carybrooke currently reside in Corinth, Maine and Gabriel is a former resident of Hampden, Maine. Gabriel graduated from Bangor Christian schools in 1997, and from the University of Maine, Orono in 2001. He has a B.A. degree in mathematics. Gabriel will begin a teaching career at Eastern Maine Community College in the Fall of 2003. He is a candidate for the Master of Arts degree in Mathematics from The University of Maine in August, 2003.