**Alethic Data**

Alethic Data

Alethic Data

Copyright © 2014 by Jason Adrian Williams

No part of this document may be reproduced in any manner without written permission from the author.

**Abstract**

Contained herein is the amalgamation of my research, Alethic Data. Logic and algebra are intrinsically intertwined in the eternal dance of reality; one cannot exist without the other. In this paper, I discuss a variety of applications and properties of the duality of logic and algebra. I begin with a brief introduction to the core structures and definitions as required by Alethic Data. After preliminary topics have been addressed, I develop some specific algebraic properties of Alethic Data. Next, I observe analytic properties inherent in Alethic Data, such as continuity, limits, derivatives, series, and integration. Finally, I conclude with applications of Alethic Data. A general knowledge of contemporary abstract algebra is recommended of the reader; however, appendix D contains a brief review of the necessary topics to follow my thesis.

Contents

[Preface 1](#_Toc477901414)

[Preliminaries 3](#_Toc477901415)

[0: Introduction 4](#_Toc477901416)

[1: The Standard Logical Universe 8](#_Toc477901417)

[2: Fundamental Properties of Logic 11](#_Toc477901418)

[3: Conversion Tensors and Operators 20](#_Toc477901419)

[4: The Logical Expansion of Polynomials 24](#_Toc477901420)

[5: Analytic Processes of Computation 30](#_Toc477901421)

[Algebraic Theory over Alethic Spaces 35](#_Toc477901422)

[6: Existence and Uniqueness of Alethic Operators 36](#_Toc477901423)

[7: The Infinite Singularity and Alethic Spaces 40](#_Toc477901424)

[8: Homomorphisms on Alethic Spaces 47](#_Toc477901425)

[9: Self-duality, Nilpotents, and the Kernel 57](#_Toc477901426)

[10: Involutions over Alethic Spaces 61](#_Toc477901427)

[11: Permutations and the Logical Search Algorithm 64](#_Toc477901428)

[Analysis of Alethic Spaces 67](#_Toc477901429)

[12: Closed Alethic Spaces 68](#_Toc477901430)

[13: Alethic Subspaces 73](#_Toc477901431)

[14: Open Extensions over an Alethic Space 77](#_Toc477901432)

[15: Continuity and Limits on Open Extensions 88](#_Toc477901433)

[16: Differentials, Infinitesimals, and Adjacency 91](#_Toc477901434)

[17: Alethic Vector Fields 95](#_Toc477901435)

[18: Sequences and Series 99](#_Toc477901436)

[19: Integration on Extension Spaces 103](#_Toc477901437)

[20: Non-Cartesian Coordinates 106](#_Toc477901438)

[Appendix A : The Labyrinth Game 110](#_Toc477901439)

[Appendix B: Geometric Influences on the Alethic Involution 116](#_Toc477901440)

[Appendix C: Polynomial Reduction 120](#_Toc477901441)

[References 125](#_Toc477901442)

# Preface

This thesis is separated into three main parts: preliminaries, algebraic properties, and finally analytic constructs. These parts are further divided into a series of chapters. In this way, I can make clearer the purpose of each section of this document.

In the first section, I introduce and discuss basic concepts of Alethic Data. Chapter 0 is an introduction to the structure of Alethic Data. The next chapter describes the standard logical universe and preludes the concept of the infinite singularity. The following chapter describes fundamental algebraic properties of logic that are inherent to Alethic Data. Chapter 3 builds the concepts of conversion tensors and operators. The chapter after that defines an algorithm to convert polynomial functions to logical “signals” in order to optimize the expansion of complicated structures. The final chapter of the first part illustrates a theoretical process to build a microprocessor that can perform the conversions between algebraic structures and logical signals.

The second section addresses algebraic properties of Alethic Data in rigorous detail. Chapter 6 defines and illustrates the concept of Alethic operators. The following chapter builds on the infinite singularity as previously described in chapter 1; additionally, defines Alethic spaces. The next chapter extends the concept of homomorphism to Alethic spaces. The chapter after that describes three algebraic structures inherent to Alethic Data. In chapter 10 I discuss a set of structures in an Alethic space that are self-dual under inversion. The final chapter of this section introduces algebraic permutations and defines the logical search algorithm.

The third section develops analytic features of Alethic Data. Chapter 12 defines different forms of closure of Alethic spaces. The next chapter describes various structures (i.e. subspaces) that can be encapsulated into larger and more complete spaces. The following chapter defines open extensions of various closures. The chapter after that describes the continuity of an open extension and limits therein. Chapter 16 develops the concept of structures that vanish at infinity and the adjacency of events. The subsequent chapter introduces the concept of an Alethic vector fields, and further develops the infinite singularity and multi-dimensional spheres. The chapter following that defines sequences and series with respect to multi-dimensional spheres. Chapter 19 extends the concept of series to integration. The final chapter of this section extends the scope of my research to non-Cartesian coordinate systems.

Following these sections are appendices that further develop topics of Alethic Data. Appendix A serves to illustrate the use of two-state logic to build a labyrinth game. Appendix B shall develop two-state logic with influences from geometry. Appendix C demonstrates the ultimate power of Alethic Data by defining polynomial reduction. Appendix D will contain a brief review of abstract algebra.

# Preliminaries

## 0: Introduction

In its infancy, Alethic Data was a simple algebraic expansion of logic whose purpose was to streamline the presentation of a logical proof. The core strategy of this technique is to directly translate a logical set of criteria into an algebraic structure, simplify the resulting expression, and map the simplified expression back onto a logical structure. Each mapping can be calculated by matrix multiplication. The transformation matrix from 2-state logic to Boolean algebra is equivalent to its inverse under modulo 2, or 2-state, computations. The idea of a transformation matrices will be addressed in a later chapter. A function that is its own inverse is called an involution, and one example is the transformation matrix under two states as mentioned above. A very useful property of an involution is that such a function applied to itself is the identity transformation. This property is evident in “exclusive or” bit encryption. A key sequence is added to a message, addition being the algebraic mapping of “exclusive or,” to obtain an encoded message; which when delivered, can only be read by another addition of the key. Anyone other than the designated recipient who could intercept the message would be unable to read it, since what they would receive would be a sum; meaningless without the key.

The initial scope of Alethic Data was to express logic algebraically. A side effect of this mindset was blindness to the idea of the reverse. Although this may seem trivial, the ramifications of such a lapse of insight were great and plenty. I spent years of research in the development of an algorithm to simplify logic, when it may have been that the algebra was even more complex. It was not until later that I decided to broaden the scope of Alethic Data to accommodate the logical treatment of algebra. For example, I initially would have transformed the logical operator “and” into an algebraic product, whereas I could represent a product as a logical vector, and in turn a logical operator, namely “and”. As a logical statement becomes more complicated, so does its algebraic transformation; however, the primary objective of the transformation remains intact. Nonetheless, the complexity of the algebraic form is minimal compared to the axiomatic approach of logic, because it lends itself more freely to simplification.

Alethic Data has since evolved into a paradigm of the duality between algebra and logic. An application of Alethic Data is data compression, wherein each element of data stores not only its value, but attributes that can be defined for that element; furthermore, these attributes need not occupy physical space in memory. A well-known use of such types of data is in hashing algorithms, in which an array is used to organize data to be retrieved efficiently. In this case, the key by which the data is stored determines an element’s location in the array, but the attribute of location takes up no additional memory. Alethic Data utilizes the concept of indirection to minimize the use of physical memory (indirection being the use of descriptors or references to attributes of data). These data points can take many forms, such as locations, lengths, frequencies, or displacements. As I have already mentioned, a hashing algorithm is an example of the use of location to store data. Lengths can be used to store information about similar or dissimilar quantities, such as the number of distinct words in a text file; each distinct word could occupy an element of an array, and the length of the array would be the same as the number of distinct words. An application of frequency could be to set a threshold for the number of occurrences of a particular element. For example, data can be partitioned into different frequency classes, each of which could be associated with a unique attribute or process. Similar to lengths, displacements can be used to store information about particular groups of data. The most important difference between lengths and displacement is that the former is accessed through a single numeric quantity, and the latter is accessed via a difference between location pointers. These are among the most fundamental uses of Alethic Data.

One of the most important structures in algebraic logic and Boolean algebra is the integers modulo 2, abbreviated as **Z**2. In **Z**2, addition is defined by the “exclusive or” operator, and multiplication is defined by the “and” operator. If 1 is taken to mean “true,” and 0 denotes “false,” then the addition and multiplication are well defined algebraically and logically, as shown in Table 1.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| P | Q | P+Q | P xor Q | P\*Q | P and Q |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 1 |

Table

Let (P or Q) be equivalent to (P xor Q) xor (P and Q). From this definition, (P or P) is equivalent to P, and (P and P) is also equivalent to P. Algebraically, this can be proved as (P or P) = (P+P) + (P\*P), and in **Z**2, this can be reduced to 0P + P = P. A similar proof can be made for the statement (P and P). By definition, the operators (or) and (and) are idempotent (P2=P) (cf. an involution (P2=1)) for all elements of **Z**2. These, among other properties of **Z**2, extend to all integers modulo a prime, as will be shown later.

So far, I have defined a critical structure, **Z**2, and introduced a few applications of Alethic Data. I will now introduce some terminology that is central to Alethic Data. The set of all elements that are defined by certain criteria is a logical universe; the criteria can be partitioned into logical domains. The dimensionality of a logical universe is equivalent to the quantity of distinct logical domains, and the Standard Logical Universe, SLU, is three-dimensional with the domains of statements, predicates, and operators. Statements are the simplest domain in the SLU, and are defined as a string of characters that may or may not have a meaning. The variables of a function are an algebraic analog to statements. Predicates are the values, attributes, or results of a statement. The value of an algebraic variable is analogous to a logical predicate. Operators are the mappings from one logical vector space into another. A function is the algebraic dual of an operator.

All in all, Alethic Data merges the concepts of logic and algebra, and provides a framework through which their duality may be exploited. Alethic Data extends the modern theory of logical algebra to include multi-state logical structures, and aims to enhance the efficiency of algebraic algorithms by introducing a logical data structure. The ultimate goal of Alethic Data is to develop software and design hardware to facilitate the decision-making processes in virtual machines, then to yield fully autonomous AI.

## 1: The Standard Logical Universe

The most fundamental logical domain of the Standard Logical Universe (SLU) is the statement; they are everything and they are nothing. Statements alone need not have an implied meaning, yet they will always in some way be present in the SLU. All that defines a statement is its existence; nothing more, nothing less. Being the first dimension of the SLU, statements lay the framework from which more complex and specialized structures, such as predicates and operators, can take form.

If the dimensionality of the SLU is restricted to one, then all statements are stripped of their meaning, and only exist for the sake of their existence. In such a universe, the two statements “potato” and “chicken” are indistinguishable, because they would both hold no meaning. This universe lends itself to the concept of nullity via the absence of meaning; however, such structures can be legitimately defined under the notion of infinity. To see this, imagine a sphere, A, with its surface extending to infinity, and place a marble at its center, O. The marble can move in any direction within the sphere; however, it need not ever leave the center. This is equivalent to saying that any point within the infinite sphere can be its center. This becomes evident when two antipodal points (cf. the north and south poles of the Earth) are placed on the sphere. These points may be infinitely far apart (as they are on opposite “poles” of the sphere), but can be made to coincide when observed through a second infinite sphere, B, enclosing the original sphere which would now be represented by a diminishing sphere, C. Furthermore, sphere A can, when represented by the diminishing sphere C, be a unique point at the center of sphere B. Notice that sphere A contains an infinite number of points, but, in sphere B, it need not occupy more than a single point. Equivalently, every point in sphere B can independently contain the space of the infinite sphere A. Another way to observe this is to compare the rate of growth between infinite spaces. Although the definition of infinity implies that such structures are unbounded, a diminishing sphere can be used to illustrate an inner infinite space that grows slower than an outer infinite space. The radius of such a sphere would diminish at a rate proportional to ratio of the outer growth rate to the inner growth rate.

Statements alone do not describe much of logic; another domain is necessary to facilitate meaning. This second dimension is the domain of predicates, which denote a value, attribute, or implication of a statement. Now, potato and chicken can both have the attribute food, and chicken can be poultry, whereas potato could be starch. Not only do predicates elicit meaning, they can describe actions or effects of a statement. For example, potatoes could grow, and chickens can lay eggs. Furthermore, trees also grow, and snakes can lay eggs. These relations imply that potatoes are similar to trees by the aspect of growth, and that chickens are similar to snakes as they both lay eggs. Also note that both chickens and snakes can grow; implying a relation among potatoes, trees, chickens, and snakes. Predicates are well known in the study of logic, so their treatment here is superfluous; however, it is important to note that predicates are a special class of statements.

In addition to statements and predicates, there exists a third logical domain, namely operators. Objects in this domain are statements that provide a mapping from a particular set of predicated statements to a particular set of states. There are two classes of operators; quantifiers and transmutations. The former include statements that assign a pseudo-numerical value to a predicated statement; such as all, some, or none. The latter are logical functions that combine statements and their predicates to derive a final value or state. Some examples of transmutations are the following: and, or, not, implies, biconditional, exclusive or, etc. Operators are a special class of predicates. The division of standard logic into these domains provides a necessary distinction between different classes of statements. Predicates are specialized to give statements attributes and behaviors, and operators are further specialized to provide a means to unify statements and their predicates in order to obtain more meaningful expressions. Logical operators behave in a similar way as a conjunction does in language; they facilitate connections between different structures.

If you recall, **Z**2 denoted the field of integers under modulo 2 arithmetic, and it is essential in the treatment of the Standard Logic Universe (SLU) which spans statements, predicates, and operators of exactly two values. By this I mean that a statement can only be true or false, a predicate can either apply to a statement or not, and an operator, as a class of statements, can only yield one of two results, namely true or false. The effect of an operator is of particular significance, since they are by far the most complex of the three domains. All three domains have the same number of values as there are elements in **Z**2; the logical values are true and false whereas the algebraic values are 1 and 0. In fact, true and false can be denoted by 1 and 0, respectively. I would like to clarify that the answer to the question of whether a predicate applies to a statement or not can be treated as true or false, respectively. Through truth, all three domains are united.

## 2: Fundamental Properties of Logic

There are few things as straightforward as truth; either the value of a statement is true, or it is false. On the other hand, how one determines whether a general statement is true or false is anything but straightforward. In this context, statements are of any of the three logical domains; the concepts of predicates and operators are simply special cases. Do not confuse this generalization with a one-dimensional logical universe, because statements under this treatment can have attributes and effects on other statements. Furthermore, these statements will have the value of either true or false, or algebraically 1 or 0. It is convenient that these are also the only elements of **Z**2, since this field has many unique properties.

Recall that an involution is a function that is its own inverse. **Z**2 has only the zero and unity elements, and under addition it forms an involution. To show this, observe that the additive identity is 0, and 0 + 0 = 0 (mod 2) and 1 + 1 = 0 (mod 2). These equations imply that 0 is the inverse of 0, and 1 is the inverse of 1; they are both their own inverses. In addition, the non-zero elements of **Z**2 form an involution under multiplication; 1 is the only such number and 1 \* 1 = 1 (1 being the multiplicative identity). This property will be extended to more complex structures in a later chapter.

Before I continue, allow me to define the fundamental operators “and” and “exclusive or” (xor). The former is algebraically represented by a product; the latter by a sum. A special case of “xor” is “not” (the statement “not p” is denoted by 1 + p). To demonstrate the application of each operator, observe their effects on two statements “p” and “q” (each of which can only have the value of 1 or 0). “p and q” is true if and only if “p” is true and “q” is also true; similarly, p \* q = 1 if and only if p = 1 and q = 1 (recall that we are working with integers modulo 2). “p xor q” is true if and only if exactly one of either “p” or “q” is true; note that p + q = 1 if and only if p = 0 and q = 1, or p = 1 and q = 0. As an extension of xor, “not p” is the same as 1 + p, since this is only true when p = 0; furthermore, if p = 1 then 1 + p = 0 (mod 2). Since these properties hold, “not p” will be denoted by 1 + p, “p xor q” by p + q, and “p and q” by p\*q (alternatively pq). Using this notation, “not (p and not q)” can be written as 1+p(1+q). Furthermore, the operator “p implies q” (false if and only if p is true and q is false), is used extensively in the treatment of logic. As you can see, “not (p and not q)” is identical to “p implies q,” since I have defined “p implies q” as implies(p,q) = 0 if and only if p = 1 and q = 0, or implies(p,q) = 1+p(1+q). Since this structure is used so widely throughout logic, I will abbreviate “p implies q” as p ~ q.

Another useful operator is the biconditional, which is an equivalence relation under logic; it has the following three properties: reflexivity, symmetry, and transitivity. The biconditional is defined as “ ‘p implies q’ and ‘q implies p,’ ” or (p~q)(q~p). A relation (R) is reflexive if pRp (p maps to itself under the relation), symmetric if pRq~qRp (“p maps to q” implies “q maps to p”), and transitive if (pRq)(qRr)~(pRr) (“ ‘p maps to q’ and ‘q maps to r’ ” implies “p maps to r”). Algebraically, the biconditional is written as:

(1 + p(1+q)) \*(1 + q(1+p)) (definition of biconditional)

(1+p+pq) \*(1+q+qp) (expanding)

(1+pq+p)\*(1+pq+q) (commutative properties of addition and multiplication)

(1+pq)2+ (p+q) (1+pq) +pq (expanding)

(1+pq)+ (p+q) (1+pq) +pq (mod 2)

(1+p+q) (1+pq) + pq (factoring)

(1+p+q) + (pq+p2q+qpq) +pq (distribution)

1+ p+q + (p2q+(1+q)pq)+pq (additive commutativity and factoring)

1+ (p+q) + (p2q)+(1+q)pq+pq (associative property of addition)

1 + (p+q) + (p2q) + q(pq) + pq + pq (commutative property of addition and distribution)

1 + p + q + pq +qpq (mod 2)

1 + p + q + (1 + q)pq (factoring)

1 + p + q (law of excluded middle) [u(1 + u) = 0]

To show that the biconditional is reflexive (see above definitions), 1+p+p = 1+0p(mod 2) = 1. As for symmetry, let 1+p+q = 1+q+p, and by the commutative property of addition, 1+p+q=1+p+q; by reflexivity, this is true.

Finally, for transitivity,

(pRq)(qRr) ~ (pRr) (definition of transitivity)

(1+p+q) (1+q+r) ~ (1+p+r) (substitution)

(1+q+p) (1+q+r) ~ (1+p+r) (commutative property of addition)

(1+q)2+(p+r)(1+q)+pr ~ (1+p+r) (expansion)

1+q + (p+r)(1+q) +pr ~ (1+p+r) (mod 2)

(1 + p + r)(1 + q) +pr ~ (1+p+r) (factoring)

1+[(1+p+r)(1+q)+pr][1 + (1+p+r)] (definition of ~)

1+[(1+p+r)(1+q)+pr][p+r] (mod 2)

1+(1+p+r)(1+q)\*(p+r)+ pr(p+r) (right distribution)

1+(p+r)\*(1+p+r)(1+q) + pr(p+r) (commutative property of \*)

1+pr(p+r) (law of excluded middle)

1+prp+pr2 (expansion and commutative property of multiplication)

1 + prp + pr (mod 2)

1 + pr(p + 1) (mod 2)

1 (law of excluded middle)

As you can see, the biconditional operator is indeed an equivalence relation; I will abbreviate “p biconditional q” with p = q. Two important points to observe are that all powers under mod 2 are linear, and that the coefficients of each term are 1 if odd or 0 if even. Furthermore, the coefficients are effectively “xor” sums, and the powers are “and” products. Obviously, this makes computations far simpler. Along with the pure algebraic approach to logic, an axiomatic simplification can be employed, namely the law of excluded middle. This is a logical property in which a statement is exclusively true or false, thus the product of a statement and its negation will always be zero (false).

Many properties of logic can be expressed algebraically:

(2.1.1) p2 = p\*p = p (p is always the same as itself)

(2.1.2) 2p = p+p =0 (p is never the opposite of itself)

(2.2.1) p\*q = pq (mod 2) (\* is a product under mod 2)

(2.2.2) p+q = q+p (commutative property of +)

(2.2.3) p\*q = q\*p (commutative property of \*)

(2.3.1) p\*(p+q) = p + p\*q = p\*(1+q) (first absorption property)

(2.3.2) p+(p\*q) = p + p\*q = p\*(1+q) (second absorption property)

(2.3.3) p\*(p+q) = p+(p\*q) (commutative property of the absorption operator)

Properties (2.1.1) and (2.1.2) illustrate the effect of doubling a statement under either operator. (2.2.1) states that the meet, \*, of two statements is a product. Since **Z**2 is a field, the join, +, and the meet are both commutative, thus (2.2.2) and (2.2.3) are implied. The absorption operators, the meet of a statement and its join to another (2.3.1) and the join of a statement and its meet with another (2.3.2), are identical (2.3.3). Furthermore, if ‘n’ is a prime number and p,q ∈ Zn, then the following hold under modulo n arithmetic:

(2.4.1) pn = p (invariance principle)

(2.4.2) np = 0 (characteristic principle)

(2.5.1) pn-1 = 1 unless p = 0 (mod n) (unity)

(2.5.2) pn-2 = p-1 unless p = 0 (mod n) (multiplicative inverse)

(2.6.1) pn-1\*(p + q) = p + q unless p = 0 (mod n) [first absorption property]

(2.6.2) p+(pn-1 \* q) = p + q unless p = 0 (mod n) [second absorption property]

(2.6.3) pn-1\*(p+q)=p+(pn-1\*q)(mod n) [commutative property of absorption]

The first group of properties (2.4.1) and (2.4.2) are just a generalization of (2.1.1) and (2.1.2), respectively. (2.5.1) and (2.5.2) are derived from (2.4.1). Finally, (2.6.1)-(2.6.3) reflect a set of general absorption properties of algebraic logic, which follow from (2.5.1). It is critical to note that the multiplicative identity of the field **Z**p is quasi-distributive (i.e. a(b+c)=b+ac). Alternatively, p (denoted as b) absorbs pn-1 (denoted as a). Furthermore, ‘a’ is bound to multiplication, and ‘b’ is bound to addition; in other words, the product commutes with a, and the sum commutes with b. Also, another distribution over the absorption operator exists, and yields the equation a(b+c) = ab+c, as is the case when ‘a’ is the unity of q (i.e. qn-1), ‘b’ is p, and ‘c’ is q. It then follows that b+ac = ab+c = a(b+c).

(T2.1.1) A unity of any prime-state logic, L, exists.

Proof: Let 1 and ‘a’ be any two elements of L. If 1a=a1=a, then 1 is a unity. L is additively cyclic with any positive element as a generator. If 1 is not a unity, then it is a generator; therefore, an integer k exists such that k1=1k=1. By the definition of unity, ‘k’ is a unity of L, and 1 remains arbitrary.

(T2.1.2) A unique multiplicative inverse exists for every non-zero element of L.

Proof: Let 1 be the multiplicative identity (unity) of L, and ‘a’ be any element of L. The multiplicative inverse of a is defined as the element ‘a-1’ such that aa-1=a-1a=1. This element is identical to ap-2 (as shown above), and since a∈L, ap-2∈L follows (a can be multiplied by itself ad infinitum to generate a cyclic subgroup of L) (p is the number of states in L). Let b’ and b be elements of L such that ab=1 and ab’=1. Since 1=1 by reflexivity, ab=ab’. Finally, we can cancel the a’s since L is a field and in turn an integral domain.

(T2.1.3) The unity of any PS logic, L, is unique.

Proof: Let 1 and 1’ be two unities of L. Allow ‘a’ to be any non-zero element of L. The two equations 1a=a1=a and 1’a=a1’=a follow from the definition of unity. Furthermore, 1a=1’a=a and a1=a1’=a; by right and left cancellation, 1=1’. Since the unity is unique, 1 will be used exclusively as the unity for all L.

Symbolically,

(T2.1.1) ∃1(1a)=a and ∃1(a1)=a,

(T2.1.2) ∃ap-2(aap-2=1) and ∃ap-2(ap-2a=1) unless a=0,

(T2.1.3) ∀a[(1a)=(1’a)=a] and ∀a[(a1)=(a1’)=a] implies 1=1’.

(T2.2.1) A zero of any PS logic, L, exists

Proof: Let 0 and ‘a’ be any two elements of L. If 0+a=a+0=a, then 0 is a zero. L is additively cyclic with any positive element as a generator. If 0 is not a zero, then it is a generator; therefore, an integer k exists such that 0+k=k+0=0. By the definition of zero, ‘k’ is a zero of L, and 0 remains arbitrary.

(T2.2.2) A unique additive inverse exists for every element of any PS logic, L.

Proof: Let 0 be the additive identity (zero) of L, and ‘a’ be any element of L. The additive inverse is defined as the element ‘-a’ such that –a+a=0 and a+(-a)=0. This element is identical to p-a.

(T2.2.3) The zero of any particular PS logic, L, is unique.

Proof: Let 0 and 0’ be two zeros of L. Allow ‘a’ to be any element of L. –a+a=0 and –a+a=0’; therefore, 0=0’. Since the zero of L is unique, 0 will be used exclusively to denote the zero of L.

Symbolically,

(T2.2.1) ∃0(0a=0) and ∃0(a0=0),

(T2.2.2) ∃(-a)(-a+a=0) and ∃(-a)(a+-a=0),

(T2.2.3) ∀a(0a=0’a) and ∀a(a0=a0’) implies 0=0’

Alone, the unity (1) and zero (0) build the structure of 2-state logic. The algebraic construct of 2-state logic is the field **Z**2, which consists exclusively of 0 and 1. The two sets of theorems (T2.1.1)-(T2.1.3) and (T2.2.1)-(T2.2.3) demonstrate the existence and uniqueness of the multiplicative and additive identities and inverses.

Any statement in L2 (2-state logic) will have the value of either 1 or 0, and the fundamental operators are “and” and “xor.” Multiple statements may have any combination of ones and zeros (i.e. The set of statements p and q have the possible state sequences (p,q)∈{(0,0),(0,1),(1,0),(1,1)} ). Operators also have state sequences, but with the order (size) of 2n where n is the number of statements (i.e. pq=(0,0,0,1), p+q=(0,1,1,0), 1+p=(1,1,0,0), 1+q=(1,0,1,0), 1+p(1+q)=(1,1,0,1)). Furthermore, operators can be applied to other operators yielding more complex structures. On the other hand, the coefficients of any term of a polynomial under modulo 2 will be 1 if present or zero if not (i.e. pq=(0,0,0,1), p+q=(0,1,1,0), 1+p=(1,0,1,0), 1+q=(1,1,0,0), 1+p+pq=(1,0,1,1)). The next two chapters will be devoted to the extension of logic in **Z**2 to logic in any finite field (i.e. **Z**p where p is prime).

## 3: Conversion Matrices and Operators

The time has come to introduce the cornerstone of Alethic Data: conversion Matrices. For each statement of the familiar 2-state logic, the matrix has the form:

|  |  |
| --- | --- |
| 1 | 1 |
| 0 | 1 |

Figure

For two statements, the matrix is:

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 |

Figure

As you can see, the matrix expands to accommodate each additional statement. Furthermore, the expansion is iterative (i.e. each cell can be calculated by multiplying corresponding entries of the fundamental matrix). The equation for a particular cell of a 2-state logic matrix is

k(t) = kth bit of t as written in binary

I = Ith row

J = Jth column

n = number of statements

MI,J = element (I, J) of transformation matrix

For example, element (2,3) can be defined in the following manner:

The matrix for 2-state logic over two statements is:

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 |

And element (2,3) is here

Recall that the inverse matrix (i.e. the transformation matrix from logic to algebra) is, for 2-state logic, the same as the matrix from algebra to logic. In other words, 2-state logic forms an involution with algebra. To compute the logical vector, L, (if the algebraic vector, A, is known) multiply A by the transformation matrix, M. Alternatively, the product LM-1 yields the algebraic vector. P-state data structures can be defined when P is prime. In the general single statement case, the cells of the transformation tensor and its inverse are:

MIJ= δ0J δI0 + JI (1 – δ0J) (mod P) (A2L)

M-1IJ= δ0J – δI0 δ(P-1)J – I-J(1 – δI0) (mod P) (L2A)

δ is the Kronecker delta function (1 if the indices are the same, 0 otherwise)

In a multi-statement environment, each statement must have the same number of states, and each cell is a product of corresponding elements of the appropriate fundamental matrix. There are two types of operators, quantifiers and transmutations. The most familiar transmutations perform on two-state logic; however, the notion of such operators can be extended to any prime-state logic. Quantifiers have two forms; the first being the absolute supremum, absolute infimum, or an assignment of the state values over a set of data under any particular prime state logic, and the second being any relational operator. There are three main quantifiers of the first form: the existential quantifier, ∃, is the absolute supremum; the universal quantifier, ∀, is the absolute infimum; and the biconditional quantifier, :, is an assignment relation over any particular prime state logic. The most critical distinction between the biconditional quantifier and the biconditional operator is that the quantifier *sets* the value of a logical or algebraic expression, and the operator *tests* the value of a logical or algebraic expression. The notion of a biconditional quantifier completes the concept of quantification along with the existential and universal quantifiers. The biconditional quantifier takes two arguments, and the result is an assignment of the right argument to the left argument. Relational operators, on the other hand, compare the values of two operands, and return one of two states; typically true or false (1 or 0, respectively). The table on the following page illustrates the use of relational operators.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| P | Q | P < Q | P > Q | P = Q | P ≤ Q | P ≥ Q | P ≠ Q |
| 1 | 3 | 1 | 0 | 0 | 1 | 0 | 1 |
| 2 | 2 | 0 | 0 | 1 | 1 | 1 | 0 |
| 5 | 1 | 0 | 1 | 0 | 0 | 1 | 1 |
| 4 | 6 | 1 | 0 | 0 | 1 | 0 | 1 |

As you can see, the application of a relational operator yields either true or false, 1 or 0 respectively. These six fundamental relational operators build an ordering structure over both logic and algebra. The following relation compares the universal, biconditional, and existential quantifiers: ∀P ≤ P:P ≤ ∃P. This relation provides boundary conditions on any prime state logic; ∀P represents the lower bound of P, P:P is the assignment of P to itself, and ∃P denotes the upper bound of P.

## 4: The Logical Expansion of Polynomials

Conversions between logical and algebraic statements are defined by multiplication of the statement structure by the transformation tensor consistent with the size of the statements and direction of conversion. Suppose it was required that a program be built that takes three algebraic input statements (two operands and an operator; each of which are represented by the coefficients of a polynomial) of equal length, P2 (P being necessarily a prime, and squared due to the quantity of operands), and produces a tensor of output results. To accomplish this, build both conversion tensors that correspond to the length of the statements (the second iteration of the P-state tensor). The input statements should be in tensor form (in this case the size of each tensor should be 1 by P2) each statement being a separate tensor. Next, multiply the statement and operator tensors by the conversion tensor, and store the results in the corresponding tensor. Then, select the operator and statements from the corresponding tensors; compute the resulting value by locating the element of the operator that corresponds to the values of the operands being processed. If the statements have the values 3, and 4 at position X, then the result at X will be the value of the operator at the element corresponding to (3,4). To return the values to algebraic form, multiply the tensor of results by the inverse conversion tensor.

Take, for example, the algebraic statements f: x2y + 2z, g: x2 + y2, and h: x + y + z and the operators F: f + 2gh + 1, G: f + g, and H: F + G2 (assume a 3-state structure). To determine the value of H, rewrite each statement as its coefficient representation:

f: 020 000 000 000 000 000 000 100 000

g: 000 000 100 000 000 000 100 000 000

h: 010 100 000 100 000 000 000 000 000

F: 100 020 000 100 000 000 000 000 000

G: 000 100 000 100 000 000 000 000 000

H: 000 000 100 100 000 000 000 000 000

Then, multiply each vector by the 3-state conversion tensor for three variables to get the following logic vectors:

fM: 021 021 021 021 102 210 021 102 210

gM: 000 111 111 111 222 222 111 222 222

hM: 012 120 201 120 201 012 201 012 120

FM: 111 102 120 222 210 201 000 021 012

GM: 000 111 222 111 222 000 222 000 111

HM: 000 111 111 111 222 222 222 000 000

Next, compute the logical structure of FM(x,y,z), GM(x,y,z), and HM(x,y,z):

FM(x,y,z): 102 012 201 012 111 000 201 222 111

GM(x,y,z): 021 102 102 102 021 102 102 021 102

HM(x,y,z): 110 110 002 110 122 101 002 200 212

Finally, multiply HM(x,y,z) by the inverse conversion tensor (M-1) to get:

H(x,y,z): 121 200 100 200 000 200 100 110 000

This translates to the following algebraic expression:

H(x,y,z): 1 + 2z + z2 + 2y + y2 + 2x + 2xy2 + x2 + x2y + x2yz

This expression is equivalent to:

P(x,y,z): ((x2y +2z) + 2(x2 + y2)(x+y+z) + 1) + ((x2y + 2z) + (x2 + y2))2 (mod 3)

Assume that a structure called TENSOR has been defined as a two dimensional array of integers, with the numbers of rows and columns stored as integers, and an integer return status to denote the direction of computation.

To translate the above procedure into a program:

1. Prompt the user for the structure of P(x,y,z) and the modulus of computation. Then, tokenize P(x,y,z) into its components (H: F+G2, F: f + 2gh + 1, G: f+g, f: x2y+2z, g: x2+y2, h: x+y+z).
2. Next, build a two-element array for the conversion tensors (M and M-1) and build an array of TENSORs C and a second array of TENSORs A to hold computations. Each element of C will hold the coefficients of one statement and will have the return status of 0. Initialize all elements of every TENSOR in A to 0.
3. After M, M-1, and C have been determined, take the right product of C and M (CM will convert each TENSOR in C to its logical equivalent) and set each return status to the value of 1.
4. When that has been completed, perform the “logical search algorithm” by taking the values of TENSORs f, g, and h at each column, locate the corresponding element of the TENSORs F and G and store the results in the TENSORs of array A corresponding to F and G. (If (f, g, h): (1, 0, 2), then the elements of F and G at (1,0,2) are recorded into the computation TENSORs corresponding to F and G).
5. Then, copy the updated TENSORs of A to the corresponding TENSORs of C (i.e. when the computations for F and G are complete, update F and G).
6. After that, repeat steps 4 and 5 with H (same process, but using the updated values of F and G in place of f, g, and h to compute H analogously).
7. Finally, to convert H back into algebraic form, multiply H by M-1 and change the return status of H back to 0.

Note that H, G, and g depend on only two variables, whereas F, f, and h depend on three. The variables absent from H, G, and g can be assumed to be 0. The only requirement is that statements cannot be recursively dependent (i.e. F:2G+f and G:F+g are incompatible).

The most complicated operations are the conversions between logic and algebra, but greater computational flexibility can be achieved by converting algebra into logic and using the above algorithm. Furthermore, the algorithm can be repeated with another prime modulus to extend the result to the product of those primes. For example, under modulo 5 the coefficients of the result may be different than those under modulo 3. If this is the case, that coefficient must simultaneously satisfy both values, and the result will be a member of the integers modulo 15. (i.e. if x = 1 (mod 3) and x = 3 (mod 5), then x = 13 (mod 15)). To determine the value of x under the product modulus, take the first 5 (the second prime) multiples of 3 (the first prime) and add the value of x under modulo 3 (the first prime) to each. Exactly one of these elements will satisfy the condition under modulo 5. This process can be repeated for other primes ad infinitum to generate large algebraic and logical structures by analyzing much smaller structures.

So far, we can express products of unique primes as their factors. Next, we can extend the scope of our analysis to powers of a single prime by allocating statements whose quantity and length are equal to the power. (i.e. a number modulo 9 can be represented under modulo 3 by 2 statements; each with coefficients of length 2). These two extensions can be combined to generate a structure of any integer modulus. One critical restriction is that the least prime modulus must exceed the degree of the expanded polynomial. (x2 + 1 cannot be evaluated modulo 2, because the power of 2 on x2 decays to 1; yielding an ambiguous structure. x2 + 1 is the same as x + 1 under modulo 2).

If a complete expansion of a polynomial expression is desired, find the coefficients of the expanded polynomial under prime modulo greater than the degree of the final polynomial. Then use the conversion algorithm to find the coefficients of the polynomial modulo a product of primes. When the product is greater than all coefficients of the expanded polynomial, the expansion is complete. To determine when this occurs, take the next iteration and compare the coefficients of the resulting polynomial with that of the previous polynomial expansion. For example, (x+1)5 has degree 1\*5, so the minimum prime that must be used is 7. Under modulo 7, (x+1)5 has expansion x5+5x4+3x3+3x2+5x+1; under modulo 11, (x+1)5 is x5+5x4+10x3+10x2+5x+1; under modulo 13, (x+1)5 is also x5+5x4+10x3+10x2+5x+1. Note that 10 is equivalent to 3 modulo 7. The expansion is complete at modulo 7\*11, because further iteration will generate the same coefficients as those of the structure determined by applying mod 7 and mod 11. In fact, the expansion is complete under modulo 11 alone since 11 exceeds all coefficients of (x+1)5.

As for P(x,y,z):((x2y +2z)+2(x2 + y2)(x+y+z)+1)+((x2y+2z)+(x2 + y2))2, determine the greatest power, 2\*2 = 4 and take the next prime, 5, as the first modulus. Then, determine the least number of variables (H:F+G2, F: f + 2gh + 1, G:f+g, f: x2y+2z , g:x2+y2, h:x+y+z. The least number greater than or equal to the number of independent variables of each statement is 3). Next, follow the above algorithm for modulo 5 and three variables. Repeat as necessary for higher primes.

## 5: Analytic Processes of Computation

Alethic Data exploits the duality of algebra and prime-state logic to optimize the analytic processes of a microprocessor. What remains to be shown is that such a microprocessor exists, or that software can be produced for microprocessors whose architecture is already in use. As demonstrated in the previous chapter, software can very well be made to run the structure and analysis of Alethic Data in any widely used programming language. The only restriction is how that language emulates the processes of computation under the paradigm of Alethic Data.

A more daunting task would be to build a microprocessor specifically to convert between logic and algebra. Since each cell of both conversion tensors can be accurately predicted for any prime state structure, the microprocessor need only have four functions: produce the value of any cell of either conversion tensor for any number of statements, hold the locations of the input (operators and their dependencies must be specified, as described below), output, and computation structures as stored in memory; perform the necessary products of inputs with the columns of the conversion tensor corresponding to the direction of conversion; perform the logical search function on all operators; finally, restore the data to its algebraic form.

Let us assume a microprocessor can be built to handle at most sixteen input and sixteen output statements per conversion, each statement being stored in a structure that consists of a one dimensional array of integers, a length variable, and a return status to denote the direction of conversion. This structure is similar to the TENSOR structure that was described in the preceding section, but consists of only one row, so this structure will be called VECTOR. Another necessary set of inputs would be the number of statements as an integer and the moduli under which the computations will be performed (let the moduli be in another VECTOR with length being the quantity of moduli and return status 0 to denote that this VECTOR contains the moduli and has no direction). Under each modulus, the microprocessor must convert the input statements between logic and algebra and store each column (as they are converted) to the computation VECTOR. When the computations for each statement are complete, update the input VECTOR corresponding to those statements and multiply their return status by –1 to denote that a conversion had been performed. Then, perform the logical search algorithm between all operators and their dependencies. Finally, return the VECTOR of the final result to its algebraic form.

To optimize the preceding procedure, the locations of each statement can be determined by its attributes. For example, the addresses (stored in an array) of the statement VECTORS can have four parts: a dependency construct, the statement’s cardinality, a value to distinguish between input, output, computation, and modular VECTORs, and a null byte. For a 32-bit microprocessor, the first sixteen bits of the address refer to the dependencies of the statement; the next four determine the cardinality of the statement, the following four denote the type of data, and the final 8 bits will all be 0. The value at that location will be the address of its corresponding VECTOR.

Another process that must be accounted for is the derivation of each cell of the conversion tensor and the product of the cell with the corresponding entry of the VECTORs to be transformed. There is no need to build the entire TENSOR, because each cell can be derived by the functions for M and M-1 as described in the section on conversion tensors. Each column of the result VECTOR can be calculated by taking the scalar product of the input VECTOR and the corresponding column of the conversion tensor by the following formulae:

LI = A∙MI and AI = L∙M-1I

Or in tensor notation:

LJ =AIMIJ and AJ=LIMIJ

In the above tensor equations, repeated indices are summed over, subscripted indices indicate covariance, and superscripted indices denote contravariance. This convention is known as Einstein summation notation. By substituting the functions for the cells of M and M-1, the formulae for a single statement structure become:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | P-1 |  |  |  |
| LJ | = | ∑ | AI | \* | (δ0JδI0+JI(1–δ0J)) |
|  |  | I=0 |  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | P-1 |  |  |  |
| AJ | = | ∑ | LI | \* | (δ0J – δI0δ(P-1)J – I-J(1 – δI0)) |
|  |  | I=0 |  |  |  |

And for multi-statement structures (let N be the number of statements):

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | PN-1 |  |  | N |  |
| LJ | = | ∑ | AI | \* | ∏ | δ0k(J)δk(I)0+k(J)k(I)(1 – δ0k(J)) |
|  |  | I=0 |  |  | k=1 |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | PN-1 |  |  | N |  |
| AJ | = | ∑ | LI | \* | ∏ | (δ0k(J) – δk(I)0δ(P-1)k(J) –k(I)-k(J)(1 – δk(I)0) |
|  |  | I=0 |  |  | k=1 |  |

k(I) = kth element of I

k(J) = kth element of J

For example:

If I=11 (or 4) and J=21 (or 7)

then MIJ = ((0)(0)+21(1-0))\*((0)(0)+11(1-0)) = 2 \* 1 = 2 (mod 3)

and M-1IJ = ((0)-(0)(0)-1-2 (1-0)) ((0) – (0)(0)-1-1(1-0))= (-1)(-1) = 1 (mod 3)

The following arrays are the conversion tensors for N=2 and P=3:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
|  | 0 | 1 | 2 | 0 | 1 | 2 | 0 | 1 | 2 |
|  | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1  M4,7 |
|  | 0 | 0 | 0 | 1 | 1 | 1 | 2 | 2 | 2 |
| M = | 0 | 0 | 0 | 0 | 1 | 2 | 0 | 2 | 1 |
|  | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 2 | 2 |
|  | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
|  | 0 | 0 | 0 | 0 | 1 | 2 | 0 | 1 | 2 |
|  | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 0 | 2 | 0 | 0 | 0 | 2 | 0 | 1 |
|  | 0 | 2 | 2 | 0 | 0 | 0 | 0 | 1 | 1 |
|  | 0 | 1 | 2 | 0 | 0 | 0 | 0 | 2 | 1 |
|  | 0 | 0 | 0 | 2 | 0 | 1 | 2 | 0 | 1  M-14,7 |
| M-1 = | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 |
|  | 0 | 0 | 0 | 0 | 2 | 1 | 0 | 2 | 1 |
|  | 0 | 0 | 0 | 1 | 0 | 2 | 2 | 0 | 1 |
|  | 0 | 0 | 0 | 0 | 2 | 2 | 0 | 1 | 1 |
|  | 0 | 0 | 0 | 0 | 1 | 2 | 0 | 2 | 1 |

To facilitate the product of the algebraic (logic) VECTOR with the (inverse) conversion tensor, determine each column of the TENSOR separately, take the inner product of each column as they are determined and the VECTOR being processed. This task can be performed through software working directly with the microprocessor.

A more important operation that must be handled by the microprocessor is the logical search algorithm, in which the logical structure of each statement is determined by the structure of their dependencies. An example of this procedure was described in the previous chapter. With all this as described, a microprocessor can be built to handle the computations necessary for Alethic Data, but software working directly with this microprocessor should be written to streamline its analytical processes.

# Algebraic Theory over Alethic Spaces

## 6: Existence and Uniqueness of Alethic Operators

A pair of important concepts of Alethic Data is the existence and the uniqueness of transformations on logical/algebraic operators. These “Alethic” operators act logically on the connection between statements and the predicates that describe them (algebraically on variables and their values) to provide recordable information. An operator must meet the following criteria to be Alethic:

* Predictability
* Reliability
* Duality or Closure

Any operator that provides a unique mapping from an input class to an output class is **predictable**. If an operator yields the same output every time it is applied over the same set of inputs, then it is **reliable**. Alethic operators must have either duality or closure. An operator is **dual** if it is a mapping from algebra or logic to logic or algebra, respectively. Alternatively, an operator is **closed** if it is a mapping from algebra or logic to algebra or logic, respectively. Note that duality is contravariant whereas closure is covariant.

A relation exists between the logical and algebraic forms of an Alethic operator via conversion tensors. The logical form of an Alethic operator can be used to simplify the evaluation of an algebraic expression, and the algebraic form can simplify a logical expression. If you recall, **Z**2 was the field of integers modulo 2, which is used extensively throughout the Standard Logic Universe. Furthermore, conversion tensors and their inverses are the same under this field, forming an involution. An algebraic operator in **Z**2 is defined by the coefficients of an algebraic expression for which evaluation yields desired data at specific points. The logical dual of the algebraic operator is defined by the values of data at specific points (values being dual to coefficients).

An important operator over Alethic operators is the tensor product, denoted by ⊗, which is performed by multiplying each element of a tensor or Alethic operator by the entire tensor or Alethic operator. The act of taking the multi-statement expansion of a conversion tensor is an example of a tensor product; furthermore, the tensor product on an Alethic operator preserves self-duality. In a later chapter, I will discuss the effects of self-duality and tensor products on involutions over Alethic spaces. Another significant set of operators is the Alethic product, defined as the external direct sum of the inner products between a vector and the column space of the image of a category of pointers. Here, the category of pointers is defined as structures that have as their value the location of a cell in a vector, matrix, or tensor. In the categorical sense, pointers are arrows from an index class to a tensor class; in a functional sense, pointers span the indices of a tensor and supply its structure. In computer programming, a pointer is defined as a variable that holds the address of some specific element of data in the computer’s memory.

To prove that the Alethic product is indeed Alethic, we must show that all properties of Alethic operators hold. Predictability holds since a well-defined function exists for the product as mentioned above. Reliability is satisfied since any specific input yields the same output for all instances of that input. As for duality, there are two Alethic products for each Alethic operator: one to convert from logic to algebra and another to convert from algebra to logic. Note that the Alethic product is not closed, as the domain and codomain are not identical (i.e. logical operator has a different meaning than the same operator in an algebraic environment). Furthermore, the type of an operator can be realized through its direction. The following are properties of Alethic products on algebraic operators, A and A’, and logical operators, L and L’:

AM=L LM-1=A

(A+A’)M=L+L’ (L+L’)M-1=A+A’

(A∘A’)M=L∘L’ (L∘L’)M-1=A∘A’

(A⊗A’)(M⊗M) = (AM⊗A’M) = L⊗L’

(L⊗L’)(M-1⊗M-1) = (LM-1⊗L’M-1) = A⊗A’

Where + is addition, ∘ is composition, ⊗ is the tensor product, and juxtaposition is the Alethic product; note that all of these operations are preserved.

If you recall, there are three types of quantifiers: universal, biconditional, and existential. The universal and existential quantifiers are dual to each other, and both have closure (i.e. ∀∀P ∈ ∀P and ∃∃P ∈ ∃P). Also, P=P’ implies ∀P=∀P’ and ∃P=∃P’, thus reliability is satisfied (the operator has the same value for all instances of equivalent input). Finally, there exists a unique mapping from a set of statements in the category of statements to the category of universal (existential) quantifiers over that set of statements (i.e. given a set of statements, there are unique maximal and minimal elements); therefore, predictability is satisfied and both quantifiers are Alethic.

Next, the biconditional quantifier, :, can be addressed in a similar manner. Closure can be described through right transitivity (P:R and Q:R imply P:Q, but P:Q and P:R do not imply Q:R). Note that P:Q:R is equivalent to P:R and Q:R but not P:Q (although it is implied), since Q need not be equivalent to R in the final expression. The distinction between implication and equivalence lies in that the former need only be valid in one direction, while the latter must be symmetric (valid in both directions). Also, P = P’ implies P:Q = P’:Q and Q = Q’ implies P:Q = P:Q’; furthermore, P = P’ and Q = Q’ taken together imply P’:Q = P:Q’; therefore, the biconditional quantifier is reliable (equivalent input yields equivalent output). Finally, predictability is satisfied since there exists a unique mapping from the category of biconditional quantification of statements to the statements themselves (assignment). Consequently, the biconditional quantifier is also Alethic. Relational quantifiers are Alethic as well, and I will leave the proof of this to the reader. Note that random variables are not Alethic, since they are not reliable.

## 7: The Infinite Singularity and Alethic Spaces

As illustrated in the section on the singularity (restriction to one dimension) of the Standard Logic Universe, infinity can be realized through the notion of zero. There, I used a pair of spheres whose radii were infinite, and I stated that the center of either sphere could be at any point within the other sphere. Also, I showed that one of the spheres can be entirely enclosed in the other (one may grow slower than the other); furthermore, I determined that one sphere could occupy a single point in the other sphere after an infinite amount of time had passed (so long as the rates of growth are not equal), and I could call that point the center. For example, the inner sphere, In, could grow at a linear rate, and the outer sphere, Out, could grow at an exponential rate. A dual inner sphere would shrink at a rate equal to the ratio of the growth rates of sphere In to sphere Out, and the size of the dual outer sphere could remain constant. There are three cases that can occur for concentric spheres: In grows faster than Out (In/Out = ∞), I grows at the same rate as Out (In/Out = 1), and In grows slower than Out (In/Out = 0).

|  |  |  |  |
| --- | --- | --- | --- |
| Ratio  Time | 0 | 1 | ∞ |
| 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 |
| ∞ | 0 | 1 | ∞ |

The cells of the above table represent the orientation of the inner sphere to the outer sphere (0: In < Out, 1: In = Out, ∞: In > Out). The time parameter indicates the stage of mutation (0: initial stage, 1: critical stage, ∞: final stage). When the ratios are 0 or 1 (at infinity), the orientations of the spheres at all three stages have the same value; furthermore, the orientations reflect the ratio between the growth rates of the spheres. However, if the ratio is ∞ (at infinity), then the inner sphere passes through the outer sphere. Since this happens, the three stages become apparent; the intermediate stage 1 occurs when the inner sphere meets the outer sphere; before that, the stage is 0, and after the spheres meet, the stage is ∞.

(T8.1.1) If S is the set of stage classes, R is the set of ratios, and P is the set of orientation classes, then there exists a unique mapping f: S x R → P such that x denotes a direct product and f(S,R) = R when R∈{0,1}, but if R = ∞, then f(S,R) = S.

(T8.1.2) f(S,R) can be denoted algebraically with logical states 0: 0, 1: 1, 2: ∞.

(T8.1.3) Theorems (T8.1.1) and (T8.1.2) are equivalent.

Proof of (T8.1.1): If the ratio R is 0, then the inner sphere will remain interior to the outer sphere, so the orientation will always be zero. If R = 1, then there is no change in the distance between the two spheres, so the orientation will reflect coincidence, or 1 (i.e. both spheres would grow at the same infinite rate). Finally, if the ratio is greater than 1, then the orientation of the inner sphere begins interior, passes through coincidence, and ends exterior (i.e. The orientations are 0, 1, and ∞, respectively).

Proof of (T8.1.2): The correspondence of the logical classes to the orientation, stage, and ratio classes is evident through infinite self-product (repeated product of an element by itself), ergo ∏ 0 = 0, ∏ 1 = 1, ∏ 2 = ∞. To build the algebraic structure of the relation of the stage and rate to the orientation, build the inverse conversion tensor for a 3-state data structure with pointers for stage and rate and a vector for the orientation. Then, using the Alethic product of the pointers with the orientation vector, the expression that conforms to the desired data structure is: f(r,s)= 2r+2r2+rs+2r2s = 2r(1+r)+rs(1+2r). I leave it to the reader to verify this.

Proof of (T8.1.3): To show that (T8.1.1) implies (T8.1.2), observe that the former implies a unique mapping exists, and that the latter describes that mapping. Conversely, (T8.1.2) denotes a function to predict the orientation of two spheres given the stage and growth rate of that system of spheres, and (T8.1.1) states that this mapping exists and is unique. Since, both theorems imply each other, they are equivalent.

Although infinite spheres provide a unique perspective on singular logic structures, a practically defined sphere lends itself to algebraic manipulation more freely. This new structure generalizes the notion of the locus of points equidistant from a single point to define multi-dimensional spheres; where a unique sphere of dimension N can be defined by a set of N+2 points such that no K points lie in a space of dimensionality less than or equal to K-2. To visualize the 1st spherical progression, take two unique points and add a third not on the line joining the first two; there exists exactly one sphere of dimension 1 (or circle) passing through all three points. To build the next spherical progression, add a new point in 3-space not in the plane on which the 1-sphere resides, forming a spherical surface in two dimensions (2-sphere). With each successive progression, the dimensionality of each point increases by 1, and a point outside of the space spanned by the previous K points is added to the set of points. Any one point that is used to define the N-sphere can be used as an initial point of a set of vectors, each terminating at any of the other points; furthermore, these vectors form a basis of a N+1 dimensional space. This implies that an N-sphere is necessarily embedded in a space with dimensionality exactly one greater than that of the spherical surface.

Alethic Data can be described in terms of multi-dimensional spheres through conversion tensors. In the case of spheres, points are known and the center remains to be found; for Alethic Data, a logical (or algebraic) structure is known and the respective dual structure must be determined. This being said, an Alethic space is a vector space equipped with an Alethic product and component-wise addition.

The equation for an N-1 dimensional sphere is:

|  |  |
| --- | --- |
| N |  |
| ∑ | (x­ij– xi0)2 = R2 |
| i=1 |  |

Which can be expanded to:

|  |  |
| --- | --- |
| N |  |
| ∑ | (xij2 – 2xijxi0 + xi02) = R2 |
| i=1 |  |

For a 1-sphere passing through points (x11,x21), (x12,x22), and (x13,x23), the following equations can be used to determine its center:

(x112 – 2x11x10 + x102) + (x212 – 2x21x20 + x202) = R2

(x122 – 2x12x10 + x102) + (x222 – 2x22x20 + x202) = R2

(x132 – 2x13x10 + x102) + (x232 – 2x23x20 + x202) = R2

Notice that the terms x102, x202, and R2 appear in each equation; furthermore, they are all constants. Therefore, the first equation can be subtracted from each of the other equations to get:

(x122 – x112) + (x222 – x212) – [2x10(x12 – x11) + 2x20(x22 – x21)] = 0

(x132 – x112) + (x232 – x212) – [2x10(x13 – x11) + 2x20(x23 – x21)] = 0

Next, set up the equations to be linear in xi0 for all i:

2(x12 – x11) \* x10 + 2(x22 – x21) \* x20 = (x122 – x112) + (x222 – x212)

2(x13 – x11) \* x10 + 2(x23 – x21) \* x20 = (x132 – x112) + (x232 – x212)

In matrix form:

|  |  |  |  |
| --- | --- | --- | --- |
|  | 2(x12 – x11) | 2(x22 – x21) |  |
|  | 2(x13 – x11) | 2(x23 – x21) |  |

|  |  |  |
| --- | --- | --- |
|  | x10 |  |
|  | x20 |  |

|  |  |  |
| --- | --- | --- |
|  | (x122 – x112) + (x222 – x212) |  |
|  | (x132 – x112) + (x232 – x212) |  |

=

To find the coordinates of the center (x10, x20):

|  |  |  |
| --- | --- | --- |
|  | (x122 – x112) + (x222 – x212) |  |
|  | (x132 – x112) + (x232 – x212) |  |

-1

|  |  |  |
| --- | --- | --- |
|  | x10 |  |
|  | x20 |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | 2(x12 – x11) | 2(x22 – x21) |  |
|  | 2(x13 – x11) | 2(x23 – x21) |  |

=

From here, you can multiply the tensors on the right as matrices. To generalize this process for spheres of higher dimensionality, I can write the cells of the conversion tensor as:

|  |
| --- |
| Mij = 2(xi(j+1) – xi1) |

And the rows of the column vector as:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | N |  |
| Aj | = | ∑ | xi(j+1)2 – xi12 |
|  |  | i=1 |  |

Finally, the coordinates of the center:

|  |
| --- |
| Cj = Mij-1Aj |

An alternative form can be determined where the points have some displacement, Dj, from the surface of the sphere. In this case, the equations for Cj and Mij-1 remain unchanged, but Aj must be rewritten as:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | n |  |
| Aj | = | ∑ | (xi(j+1)2-xi12) – (D(j+1) – D1) |
|  |  | i=1 |  |

And the form of the sphere becomes the following:

|  |  |  |  |
| --- | --- | --- | --- |
| n |  |  |  |
| ∑ | (xij – xi0)2 | = | R2 + Dj |
| i = 1 |  |  |  |

The radius of either of these types of spheres can easily be found by substituting the coordinates of one of the points, the coordinates of the center, and the offset if necessary; finally, solving for R. A specific case in which must be addressed is that where the offset of at least one point is greater than the distance between that point and center of the sphere. In this case the right side of the equation will be larger than the left for all real values of R; however, complex valued R can be used to mitigate this contradiction. Furthermore, any such radius would not have a real part, for if it did, the square of the radius would have an imaginary part; which cannot be the difference of the distances from any point to the center or their displacement from the surface of the sphere, both being real numbers. To build an Alethic product, the points must be chosen to determine the inverse of the proper conversion tensor and the displacements must result in the components of the logical (algebraic) vector, as shown below (with the origin being the zero vector and on the surface of the sphere

|  |  |  |
| --- | --- | --- |
| ∏Mij = ½ ∏ [δ0j – δi0 δ(P-1)j – i-j(1- δi0)] | (mod P) | (A2L) |
| ∏Mij = ½ ∏ [δ0j δi0 + ji(1- δ0j)] | (mod P) | (L2A) |

|  |
| --- |
| Di = Σ (∏ Mij)2 – ti |

i is the point index (rows) and j is the variable index (columns); both starting at 0.

## 8: Homomorphisms on Alethic Spaces

Now that I have defined Alethic products, quantifiers, and operators, I can address Alethic spaces in rigorous detail. To begin, the homomorphism functor is defined as a mapping from a source category to a destination category that preserves sums, products, and identities. The tensor product of a pair of Alethic products is also an Alethic product; furthermore, tensor products satisfy distributivity over addition. For a homomorphism to be Alethic, it must be an isomorphism (the homomorphism denotes equivalence between the domain and codomain, thus has duality) or an endomorphism (the domain and codomain of the homomorphism are identical, thus has closure); otherwise, closure and duality are not guaranteed. Note that identical and equivalent are not quite the same. The former requires that the domain and codomain coincide exactly, while the latter simply implies some bijective relation between the domain and codomain.

In a logical environment, an Alethic space contains information about some class of objects. For example, I can describe the attributes of any object (such as their shape, color, or position at a specific time) each of which can be measured and quantified. I can even represent any organism as its taxonomic classification, wherein their categorical blueprint is quantifiable and ordered. One such order is alphabetical, another being by type or category, and yet another could be the organism’s habitat or its location on the food chain, etc. As I am sure you can see, there are many ways to describe the same object; all of which are quantifiable. These quantifications map directly to an algebraic analogue of the logical meaning of any attribute of the object being described. More importantly, attributes of a set of objects can be combined and can have actions on each object in that set. These combinations and actions are algebraically defined as operators and homomorphisms. In this sense, homomorphisms are generalizations of operators, and need not be closed or dual; therefore, they generally are not Alethic. For example, an action may influence an object in a way that has not been included in the set of descriptors, yet may be recorded and analyzed; furthermore, the descriptors themselves may be different on each object. For instance, I can take a chair, a table, and a book as my objects, and by analyzing particular attributes of each object I can determine if some one could be reading the book. Given specific information about the book, table, and chair, I can even estimate the age of the reader (if one exists). The details and techniques of this process are beyond the scope of this volume and will not be addressed here. However, it suffices to say that the set {table, chair, book} has properties that can be measured and new information can be determined via these properties (i.e. the reader’s existence and age). Another way to look at this is to observe a map from the input set {table, chair, book} to the output set {existence, age} as a homomorphism.

If you recall, an endomorphism is an Alethic homomorphism. An endomorphism maps elements of any category to other elements of the same category. For example, any function that transforms an algebraic (logical) vector to another algebraic (logical) vector is an endomorphism. An endomorphic extension maps a pair of dissimilar categories to a pair of identical categories. I will address endomorphic extensions in detail later.

Let φ be the conversion isomorphism, fi be an algebraic endomorphism, gi be a logical endomorphism, A be the category of algebraic operators, L be the category of logical operators, and k be a scalar. Then, the following properties hold:

(8.1.1) φ(f1 + f2) = φ(f1) + φ(f2) φ-1(g1 + g2) = φ-1(g1) + φ-1(g2)

(8.1.2) φ(kf1) = kφ(f1) φ-1(kg1) = kφ-1(g1)

(8.2.1) gi(L) = φfi(A) = giφ(A) fi(A) = φ-1gi(L) = fiφ-1(L)

(8.3.1) φ-1giφ = fi φfiφ-1 = gi

Finally, the following diagram is commutative with exact rows:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | fi |  |  |  |
| 0 | → | Ker fi | → | A | → | Im fi | → | 0 |
|  |  | φ-1⇅φ |  | φ-1⇅φ |  | φ-1⇅φ |  |  |
| 0 | → | Ker gi | → | L | → | Im gi | → | 0 |
|  |  |  |  |  | gi |  |  |  |

Note that the implied product is composition. The top row represents algebra-like categories and the bottom row represents logic-like categories. The isomorphism φ converts algebra-like categories to logic-like categories, and φ-1 is its inverse. Ker fi represents the act of not applying the functor fi, and Im fi denotes the result of applying fi to A. The categories in the bottom row can be defined dually, replacing fi with gi and A with L. The functors fi and gi must satisfy the equalities of (8.3.1); furthermore, the entire diagram represents each step of the evaluation of an expression

Since each element of either conversion tensor can be determined at will, the use of pointers to a fundamental tensor (as would operate on a single statement) would reduce the amount of space required to process the conversions. This is particularly useful if the algebraic expression requires many variables, since each variable can be represented by a pointer rather than using a very large array. For example, the conversion tensor for 5 statements under modulus 5 would require 9,765,625 entries, whereas the fundamental tensor would only require 25. There would be 5 pointers each of which would consist of a row index and a column index. Take the following as an example of using pointers and category theory:

Let F=(1+x2)(2+x+x3)+(1+x+3x2)2 (mod 7) be an algebraic expression. F can then be separated into F= Imf0= Imf1+Imf2, and further Imf1= Imf3\*Imf4 and Imf2=(Imf5)2, and again Imf3= 1+x2, Imf4= 2+x+x3, and Imf5= 1+x+3x2. F can be determined directly by expanding each of its terms algebraically. Categorically, this would appear as F=Imf0, which is simply the result of applying all functors fi.

Certainly, F can be calculated through algebraic functions alone; however, computations can be processed more quickly by introducing a logical map for each function. In this sense, F would first be converted to its logical equivalent G. Categorically, this would appear as G=Img0; furthermore, just as Imf0 can be separated into sub-functions, Img0 can as well. Also, f0 can be written as φ-1g0φ (i.e. f0 is identical to first applying φ to map the category A to its logical form in L, then g0 to perform all logical operations, and finally φ-1 to return the result to an algebraic form). Since f0 and g0 are both composite functors (i.e. functors of functors), they can be rewritten and expanded by their components. The size of a fundamental conversion tensor in either direction is the square of the modulus, and since each image depends on at most two other images, there must be two pointers to either conversion tensor for each image. The following is a demonstration of using pointers to expand F=(1+x2)(2+x+x3)+(1+x+3x2)2 in modulo 7:

First, break F into its components:

F= Im f0 = Im f1 + Im f2,

Im f1= Im f3\* Im f4,

Im f2 = (Im f5)2

Im f3 = 1 + x2

Im f4 = 2 + x + x3

Im f5 = 1 + x + 3x2

Notice that each expression depends on at most 2 variables; therefore two pointers to the conversion tensor under modulo 7, φ7, can be used to expand F.

φ7 is the following matrix:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 0 | 1 | 4 | 2 | 2 | 4 | 1 |
| 0 | 1 | 1 | 6 | 1 | 6 | 6 |
| 0 | 1 | 2 | 4 | 4 | 2 | 1 |
| 0 | 1 | 4 | 5 | 2 | 3 | 6 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 |

The algebraic vectors corresponding to each Im fi are the following:

|  |  |
| --- | --- |
| Im f0 | Im f2 |
| Im f1 | 0 1 0 0 0 0 0  1 0 0 0 0 0 0  0 0 0 0 0 0 0  0 0 0 0 0 0 0  0 0 0 0 0 0 0  0 0 0 0 0 0 0  0 0 0 0 0 0 0 |

|  |  |
| --- | --- |
| Im f1 | Im f4 |
| Im f3 | 0 0 0 0 0 0 0  0 1 0 0 0 0 0  0 0 0 0 0 0 0  0 0 0 0 0 0 0  0 0 0 0 0 0 0  0 0 0 0 0 0 0  0 0 0 0 0 0 0 |

|  |  |
| --- | --- |
| Im f2 | Im f5 |
| 1 | 0 0 1 0 0 0 0 |

|  |  |
| --- | --- |
| Im f3 | x |
| 1 | 1 0 1 0 0 0 0 |

|  |  |
| --- | --- |
| Im f4 | x |
| 1 | 2 1 0 1 0 0 0 |

|  |  |
| --- | --- |
| Im f5 | x |
| 1 | 1 1 3 0 0 0 0 |

To convert each Im fi into their logical form, Im gi, assign pointers to φ 7 from each dependency of the particular fi being converted as shown below.

The pointers begin at the upper left cell of the conversion tensor:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 0 | 1 | 4 | 2 | 2 | 4 | 1 |
| 0 | 1 | 1 | 6 | 1 | 6 | 6 |
| 0 | 1 | 2 | 4 | 4 | 2 | 1 |
| 0 | 1 | 4 | 5 | 2 | 3 | 6 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 |

Im f0

Im f2

Im f1

Take the product of all elements of φ7 that are referred to by a pointer and the element of the parent vector corresponding to the location of the pointers. In this first step the elements being pointed to have the values 1 and 1. The location of the pointer from Im f1 is 0, as is from f2; therefore, the cell of Im f0 that must be considered is at position (0,0). The product of the elements of φ7 and the cell is 1 \* 1 \* 0, which becomes 0. Next, increment the location of the first pointer by moving it down one row, and leave the other pointers where they are; as in the following diagram:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 0 | 1 | 4 | 2 | 2 | 4 | 1 |
| 0 | 1 | 1 | 6 | 1 | 6 | 6 |
| 0 | 1 | 2 | 4 | 4 | 2 | 1 |
| 0 | 1 | 4 | 5 | 2 | 3 | 6 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 |

Im f2

Im f0

Im f1

Then, take the product of the new entries and the corresponding element of f0. Think of the locations of each pointer as a digit of a number in a base equal to the modulus, in this case 7. The leftmost digit would refer to the slowest pointer, Im f1, and the rightmost digit would be the fastest pointer, Im f2. All pointers to φ7 have two coordinates (row, col): the column of the source vector determines ‘row,’ and the column of the destination vector determines ‘col.’ When a column has been computed (all pointers are at the last element of the column), add each product and place the result in the corresponding location of the destination vector. Multiple pointers can be used in an analogous manner. For example, 102 means that the slowest pointer is at location 1, the next slowest is at location 0, and the fastest pointer is at location 2. Each column of the source/destination vector can be denoted by an integer in base 7 (i.e. 102 is the 51st column of the source/destination vector). By hand, this process is tedious, but a computer can easily be programmed to perform the necessary calculations; furthermore specific elements of data can be accessed very quickly (such as a particular logical state at some location or a coefficient of some term of a polynomial).

## 9: Self-duality, Nilpotents, and the Kernel

As illustrated in the previous chapter, the homomorphism functor allows some action to occur between a source category and a destination category, and through this action determine some quantifiable result. Some elements of the source category can map to themselves in the destination category under the Alethic product; such elements are called self-dual. Some elements of the source category may map to zero in the destination category under some homomorphism, forming the kernel of that homomorphism. Furthermore, all self-dual elements belong to the kernel of the difference homomorphism (i.e. F = S – D ∈ K, where F is the difference function, S is a self-dual element of the source category, D is the corresponding element of the destination category, and K is the kernel). An important feature of the kernel of Alethic operators is that the least significant element, LSE, of those operators is in the kernel of the underlying field. Furthermore, that element remains unchanged under the Alethic product (i.e. the LSE of the difference of any algebraic vector and its corresponding logic vector is necessarily a member of the kernel of the field). Another use of the kernel regards nilpotent elements (being defined as the elements that after repeated application to themselves yield an element of the kernel). All self-dual elements of an Alethic operator can determined by using the following equation:

AM = A (equivalently, LM-1=L)

Which can be rewritten as:

A(M – I) = 0 (equivalently, L(M-1 – I) = 0)

The following are properties of the additive group of self-dual elements, S, over the ring of Alethic operators, A, with kernel, K, underlying field, F, and conversion tensor, M:

(9.1.1) 0a is self-dual for all 0 ∈ K and a ∈ A

(9.1.2) s, s0 ∈ S implies s – s0, s + s0 ∈ S

(9.1.3) ks0 = s for some scalar k ∈ F

(9.1.4) sM = s for any s ∈ S

Furthermore, if S is the group of self-dual operators, A is the category of algebraic operators, L is the category of logical operators, M is the conversion tensor, and ⊗ is the tensor product, then the following are true:

(9.2.1a) (S⊗A)(M⊗M) = (SM)⊗(AM) = S⊗L

(9.2.1b) (A⊗S)(M⊗M) = (AM)⊗(SM) = L⊗S

(9.3.1a) (S⊗L)(M-1⊗M-1) = (SM-1)⊗(LM-1) = S⊗A

(9.3.1b) (L⊗S)(M-1⊗M-1) = (LM-1)⊗(SM-1) = A⊗S

These equations above show the actions of tensor products over Alethic products (denoted by juxtaposition of tensors). Also note that the dimensionality of each tensor is as the following: M, and M-1 are of dimensionality 2 (matrix-like), and A, S, and L are of dimensionality 1 (vector-like).

Nilpotent elements are very important since they vanish after being applied to themselves some finite number of times, regardless of the order of the underlying field. For example, under one logical statement in mod 5, (02023) is nilpotent since by using the logical search algorithm, (02023)(02023)(02023) = (02023)(00002) = (00000) or (02023)3 ∈ Kerl, where Kerl is the logical kernel. Furthermore, the algebraic dual of a logical nilpotent is also nilpotent with respect to the algebraic kernel (i.e. the dual of (02023) is (02203) and (02203)3 ∈ Kera, where Kera is the algebraic kernel). Note that the product in both forms is function composition. To illustrate the advantages of the logical form of an algebraic structure consider the following:

|  |  |  |
| --- | --- | --- |
| Algebra (mod 5) | Logic (mod 5) | Alt Algebra (mod 5) |
| 2x+2x2+3x4 | (02023) | 2x+2x2+3x4 |
| 2(2x+2x2+3x4)+2(2x+2x2+3x4)2+3(2x+2x2+3x4)4 | (00002) | 2x+3x2+2x3+3x4 |
| Too long to show! | (00000) | 0 |

It should be readily discernible that the intermediate logical form can be used to build the compact algebraic structure of a nilpotent, and simplify algebraic computations. The kernel of a functor can be defined as the category of elements that map to the zero category under that functor. For a logical statement to be nilpotent, all elements of the statement map to the kernel after some finite number of iterations and remain in the kernel (i.e. the statement 02013 can follow the path 02013 to 00021 to 00000 and is therefore nilpotent). There is no general multivariate analog of the nilpotent, although there may be special cases in which a set of non-kernel elements may vanish under a specific operator; such an operator is nil over that set of elements. To illustrate a nil operator examine the following vectors:

O: 100 010 001 P: 000 111 222 Q: 121 200 110

O(P,Q): 000 000 000

O is the operator, P is the first statement, Q is the second statement, and O(P,Q) is the result of taking the operator, O, over the statements, P and Q. Individually, O, P, and Q are not members of the kernel, but the result O(P,Q) is in the kernel. Nilpotents and nil operators play a critical role in the analysis of an Alethic space by building bridges between different algebraic structures. The act of performing an operation on a set of logical statements shall be called binding, and statements are then bound to an operator; this term will be used throughout the remainder of this volume.

## 10: Involutions over Alethic Spaces

If you recall, an involution is a structure that is its own inverse. Self-dual operators are involutions under conversion. In the case of conversion tensors themselves, the only involution is under the Standard Logic Universe (SLU), where statements, predicates, and operators have scalars from Z2 only. Also, recall that two element groups are necessarily involutions, since such a group must contain an identity, e, which is idempotent (e\*e = e), and a second element, k ≠ e, such that k2 = e, since the order of any member of a group must divide the order of the group itself, and k is not equal to the identity. The field Z2 forms an involution under addition, 0+0 = 0, 1+1 = 0; furthermore this field forms a trivial involution (identity is the only member) over non-zero elements under multiplication, 1\*1 = 1. Note that 0 and 1 are the additive and multiplicative identities, respectively. The most important property of the SLU is that the conversion tensor and its inverse are the same, and can therefore be used to convert in both directions.

At this point, I shall address properties specific to **Z**2[x] (the ring of polynomials with coefficients in **Z**2), and in turn the logical form of such structures. Allow me to begin with a topic from the previous chapter: self-duality, which is a special type of involution (i.e. the inverse is taken under conversion). These elements under two statements are 0, 1, 6, and 7 which translate to “zero,” “and,” “xor,” and “or,” respectively. Note that “not” is the same as 1 xor P, where P is some single statement. Under three statements, the self-dual structures are 00, 01, 06, 07; 12, 13, 14, 15; 6A, 6B, 6C, 6D; 78, 79, 7E, and 7F. Furthermore, if the left half of any three-statement self-dual is added to the right half, the result is the converted form of the right half (i.e. 6A is self- dual, so 6 + A = C or in binary 0110 + 1010 = 1100). This property is unique to two state structures, and can be extended ad infinitum to generate large self-dual structures, which could contain copious amounts of information (i.e. a 32-bit integer is large enough to contain a 5 statement structure under two states, where each statement can operate independently). Furthermore, the fundamental operators are binary (i.e. they take two operators) and occupy exactly four elements of data (i.e. 1001 is a binary Alethic operator, namely “biconditional”). Also, pairs of operators can be concatenated to generate operators that are conditional on at least one of the statements described. For example, 10010111 depends on three statements and can be written as (1+P)(1+R+Q)+P(R+Q+QR) where P refers to a decision and the operators (1+R+Q) and (R+Q+QR) are branches that can be taken based on the result of the decision. In other words, a 32-bit integer can hold two 16-bit options, four 8-bit options, or eight 4-bit options; these refer to 4-, 3- or 2-statement structures, respectively. To facilitate multiple branches, selection variables can be used. For example, P is a selection variable in the example above (i.e. if P = 0, the left operator is used, and if P = 1, the right operator is used). To extend selection to four or eight options, use a total of two or three selection statements, respectively. Furthermore, by sacrificing one dimension, all operators can be made self-dual; however, it is important to recall that all properties in this chapter are exclusive to two state logic with algebraic expressions in **Z**2.

To illustrate how this works, observe the following:

10101100 represents a three-statement structure

00010011 is self-dual and can be used in a two-statement structure

The latter of these two examples can still be used for three statements. However, the left half of self-dual operators typically contains information about the right half of the operator. In particular, the sum (mod 2) of the left and right halves of the operator is the dual of the right half. Basically, such a structure holds the algebraic and logical forms of the operator simultaneously.

## 11: Permutations and the Logical Search Algorithm

A permutation is a bijective endomorphism, or automorphism, on a finite set of non-negative integers (i.e. in cycle notation (01)(234) is the permutation [10342]). Any book on abstract algebra will address permutations in detail; however, one may note that the index typically runs from 1 to the dimensionality of the permutation (whereas I am using **Z**p as the index). This discrepancy is immaterial as both notations are isomorphic (i.e. ∃τ : τ(x) = x-1). A k-cycle is a closed cycle of k elements (i.e. (01) is a 2-cycle and (234) is a 3-cycle). Cycle notation bears its name, because this notation lists the cycles of the permutation. Note that all 2-cycles are involutions, and all permutations can be represented as a product of involutions. Products of permutations are defined by composition and the algorithm is the following:

1. Start at the leftmost element of the right most cycle (in my example, 2 is that element). Begin the cycle.
2. Then, move one element to the right (in my example, 3 is that element).
3. Next, search for the new element in the cycles on the left, starting with the cycle just to the left of the current cycle and searching from right to left.
4. Repeat steps 2 and 3 until there are no more instances of the current element (as of the end of step 3).
5. Add the current (after step 4 has finished) element to the cycle.
6. Repeat steps 2-5 until the first element of the cycle is reached again.
7. Close the cycle.
8. Repeat steps 1-7 until all elements have been placed in a cycle.

Any permutation can be represented by a disjoint product of cycles; furthermore, any product of cycles with at least one element in common can be rewritten as a disjoint product. For example, (021)(034) can be rewritten as (03421). The former is [20134] composed with [31240] (which can be rewritten as the permutation [30142] and when written in cycle notation is (03421)). Square brackets denote the permutation by index (i.e. the leftmost element is the image of 0, and each element to its right is the image of its position), and round brackets denote the permutation in cycle notation.

To see permutations in a logical context, observe the permutation [31240] as a logical vector. This can be multiplied by the conversion tensor for one statement under five options to become the algebraic vector 30030, which is defined by the expression 3+3x3. Furthermore, the logical vector [31240] is equivalent to 3 \* [12430] (mod 5) (i.e. 3\*1=3, 3\*2=1, etc.). In fact, the expression 1+x3 can be factored as (1 + x)(1 – x + x2), or in modulo 5, (1+x)\*(1+4x+x2). Algebraically, this can be represented by f=uv, where u=1+x and v=1+4x+x2. Logically, the following:

fM(u,v) = 00000 01234 02413 03142 04321

uM(x) = 12340

vM(x) = 11323

fM(x) = 12430

uM(x) and vM(x) both depend only on x; therefore, fM(x) also only depends on x; however, fM(u,v) depends on both u and v. For this reason, fM(u,v) requires a two-statement vector, while uM(x), vM(x), and fM(x) only require one-statement vectors. Furthermore, if u depended on x and v depended on y, then f would depend on both x and y, and two-statement vectors would be required for all operators u, v, and f. The algorithm for the logical search function is similar to the algorithm of the product of permutations. For k statements nested at least once, it is the following:

1. Take each column of the matrix whose rows are the logical representations of each sub-statement, and locate the corresponding element of the logical vector of the parent function.
2. Repeat step 1 for each level of nesting.

The example above should illustrate the benefit of the duality of logic and algebra. It is not obvious that [31240] = 3(1+x)(1+4x+x2) . However, when logic and algebra are used together, this becomes evident. Note that [31240] is (034) in cycle notation. Furthermore, all permutations can be written logically and algebraically, although the algebraic expressions may not at first be obvious (logical notation is the same as index notation).

I recommend reference [1.1] pages 94-111 for a detailed discussion of permutations.

# Analysis of Alethic Spaces

## 12: Closed Alethic Spaces

Among the fundamental structures of Alethic Data are closed Alethic spaces. As we have seen up to this point, prime state logic forms a finite structure and is closed under Alethic operators. An Alethic space is a vector space with an Alethic product (i.e. conversion tensor), binding algorithm, and a logical search algorithm, and shall be denoted as *A*(VF, \*, *B*, *L*); where VF is a vector space over the field F, \* is the Alethic product, *B* is the binding algorithm, and *L* is the logical search algorithm. If all logical operators are compatible with *L* (i.e. logical operators can be composed by performing *L*), and all algebraic vectors are compatible with *B* (i.e. each algebraic vector can be bound to a logical operator), then *A*(VF, \*, *B*, *L*) is closed. For this to have any meaning, *L* shall be defined by the result of performing the logical search algorithm on a set of logical operators. The binding algorithm shall be defined as the assignment of a logical operator to an algebraic vector. Note that \* represents the Alethic product in the appropriate direction (i.e. with respect to the type of data being manipulated). This implies that an Alethic space depends on the context of the space; furthermore, a dual space exists for each Alethic space such that its context is reversed. Context shall be defined as the direction of the Alethic space (i.e. algebra to logic or logic to algebra).

An Alethic space is globally closed if *L* is generated from only the operators defined by *B* (i.e. the logical search algorithm can be performed without introducing any element not in VF). If an Alethic space is not globally closed, it can be either locally closed or open. A locally closed Alethic space is defined in the vicinity of each element of a set of points in the vector space VF. An example of a locally closed Alethic space is defined on an infinitesimal scale about any point on a differential manifold (for example, an n-dimensional sphere).

Let us examine globally closed Alethic spaces a bit more before continuing with more complicated structures. The spaces that I have addressed in the previous chapters have all been globally closed. The binding algorithm associates each point in the vector field with a specific logical statement, and the logical search algorithm computes the result of a logical operator on a set of logical statements. Im *B* denotes the category of all logical statements defined under VF, and Im *L* is the category of all results, outcomes, or predicates (i.e. the meanings or attributes of a set of statements, which are in of themselves statements) under VF. A transmutation (i.e. the result of applying a logical operator) is a special type of attribute defined by its action. In prime state logic, each statement can be considered as a variable, predicates can be regarded as the numerical state of a particular statement, and operators can be defined as functions on the set of variables. In the structures presented in the preceding chapters, the functions have been polynomials and their results are restricted to the finite field Zp. This restriction is inherent to the condition of global closure.

The main difference between global and local closure is the area of influence of each operator in either closure. The unique necessary and sufficient condition for a closure to be global is that every vector in VF is, as was specified previously, compatible with binding and logical search. However, local closure requires only that a binding algorithm exists for each element of VF; not all elements of a local closure must follow a logical search algorithm. In a later chapter, I shall discuss open extensions of locally closed Alethic spaces. But for now, just consider them as spaces that require information outside VF such as extra predicates or additional variables.

An important type of local closure consists of a set of points in n-dimensions and a set of motions on those points. On an infinitesimal scale, the progression of motion can be modeled as a set of vectors about each point. These vectors have as their magnitude the instantaneous rate of change of position at that point, and are tangent to the direction of motion. Furthermore, these vectors form a differential field about each point, and I will address the distinction between infinitesimals and differentials in a later chapter. However, it suffices to say that the intuitive model of differentiation remains intact.

In a logical setting, each point that is defined can have an implied algebraic interpretation. For example, I can take the four cardinal directions (up, left, right, and down) as different powers of two (since each direction may or may not be accessible from each point). Let up be 8 (1000), left be 4 (0100), right be 2 (0010), and down be 1 (0001). Each point defined can have any number of these accessible directions, and their notation shall be a sum (i.e. 1001 means that the directions that are accessible at the particular point are up and down; (1000) and (0001)). In this way, a two dimensional grid of points can be described (i.e. each point can be connected to any adjacent point to form a directed graph, and these connections exist whenever a motion can be accessed). By encoding each direction as an integer, we establish coordination between the values defined and their effects on each other. Note that plausible enumeration could simply be the set {0, 1, 2, 3}; however, this set does not easily lend itself to combination. For example, the sum of two or more directions cannot be expressed explicitly by these values; but by using powers of 2, a distinction can be made between each direction and they can easily be combined. The purpose of denoting directions in this manner is to isolate each motion and allow for a more general notion of combination.

With this distinction in mind, consider a finite set of events such that no two can happen simultaneously. Take for example the position of two billiard balls at some given instant of time. Since both measurements will be taken simultaneously, and no pair of billiard balls can have identical positions at any given moment; it will be unnecessary to separate any pair of such events. Therefore, the width (i.e. number of simultaneous descriptors) of any vector in a logical interpretation of these events need only be one (cf. there are four directions from the previous example, all of which may happen at once, thus the width of any such event is exactly four). In either case, the maximum of simultaneous occurrences can be categorized as the width of a predicated statement (i.e. a statement with enumeration is predicated by its attributes as defined by the enumeration). One enumeration of the billiard balls can be the number written on the ball, or zero for the cue ball; note that there are a total of 16 balls in a game of billiards. However, the modulus of a finite closed Alethic space must be prime; therefore, an extra value must be introduced as a null result. This 17th state need not ever be occupied, but it can be used to denote some set of data incompatible with the other 16 states. Alternatively, the width of the predicated statement (i.e. the set of balls) could be four, and each of the four elements can have one of two values. This second enumeration has exactly 16 members and is defined under modulo 2 as a finite closed Alethic space.

As you should be able to see, the width of a statement can not only be used to separate different elements but also to maximize the efficiency of an enumeration by reducing the modulus under which computations must be taken. Furthermore, the width itself can be used to quantify elements of data. Take for example, an algorithm to locate all five-letter words in a text document. This enumeration could be defined by an integer in base 26 (or the prime 29 to satisfy closure), and the length of the word being represented would determine the width of this integer. Words of length 5 would be encoded as integers between 294 and 295-1 under a closed Alethic space, and the width would be the larger of the two exponents (i.e. 5). Width and value are two ways that an Alethic space can be expressed, which brings us to our next topic.

## 13: Alethic Subspaces

To illustrate the notion of an Alethic subspace, I shall begin with an example. Let *A*(VF, \*, *B*, *L*) be an Alethic space with F being the integers modulo a prime. Say that *A* represents a game of billiards. We can define *B* as the binding algorithm that maps each ball to an integer. As was shown in the previous chapter, there are two obvious enumerations; we shall choose the second. *L* shall represent an algorithm to perform composition (i.e. logical search). We choose an enumeration to express the game of billiards in the most compact and direct manner available (i.e. a statement of width 4 and modulo 2). In this way, each ball could be represented by one of 16 choices or predicates. The aspect of an Alethic subspace is reflected by its context; as an Alethic space can consist of many components that exist only in their own structure. In a game of billiards, each ball can be a logical statement and be one of 24 elements. Since 24 is equal to 16, I can represent each billiard ball as a number in hexadecimal. Take the cue ball as 0, and every other ball by its number (i.e. the number on that ball). I can add any necessary parameters to the Alethic space that describes the game of billiards to whatever sophistication I desire. Each parameter can in its own right be an Alethic space, but when used together to define the game of billiards, they become subspaces of the underlying vector space.

Another example of an Alethic subspace is the quotient ring *O/M* of an arbitrary integral domain *O* with *M* being a maximal ideal in *O*. Since *M* is maximal, *O/M* is a field. Therefore, each element of *O/M* can be mapped to an element of some field *K*, which shall be the predicates of a closed set of logical statements. Furthermore, *O/M* is isomorphic to the algebraic representations of these predicates. Note that the predicates can apply to either a statement, or an operator. If *O/M* is the set of polynomials *K*[xi], then the values of *K*[xi] are in *K* (with xi a member of *K* for all i). The homomorphism *B*:*O/M*→*K* shall be the binding algorithm of an Alethic subspace of *O*. Composition of two elements in *K* shall be determined by a logical search algorithm. The condition of *K* being a field is inherent to the invertibility of each element of *K* (i.e. each element can be converted between logic and algebra by an Alethic product). The vector field VF shall be determined by *K*[xi].

As you should notice, a vector field, VF, can determine a set of logical statements, since each logical statement can be encoded as a separate element of that vector field. The binding algorithm shall be determined by a homomorphism between a logical structure, *K*, and an algebraic structure, *O/M*. Furthermore, a logical search algorithm can be defined to compose logical structures. Finally, an Alethic product can be defined to convert between logic and algebra. The overarching principle is that any logical structure can be represented by some element of a field. An Alethic subspace has all properties of an Alethic space as defined by *A*(VF, \*, *B, L)*.

In the previous chapter, I introduced the notion of the width of a statement, which can be an Alethic subspace. There, I took the width to be either 1 or 4 depending on the size of the field of operators (i.e. a width of 1 could be used if an extra case can be included, or it could be 4 if it is required that there are exactly 16 logical statements). These different widths can occupy separate Alethic subspaces of a game of billiards; furthermore, each can have its own binding algorithm. With width 1, I can introduce an error-checking algorithm, since there are 17 values (0 thru 16) for 16 balls. Each ball can be bound to the number of the ball, with the cue ball being 0. This leaves the state 16 empty; therefore, that state can be used represent any predicate. Furthermore, I can add any number of empty states as long as the total number of states is prime.

Alternatively, by using a base 2 enumeration of width 4, I can represent all 16 balls in the fewest number of states. Here, each ball can be represented by its number in base 2, and the cue ball can be 0000. For example, 1000 can represent the eight ball. Furthermore, I can enumerate the set of balls with the leading bit reflecting whether each ball is striped or solid. In this case the enumeration will be split into two categories with two null states (0000 is the cue ball and 1000 is the eight ball as before). However, the rest of the balls can lead with 1 if striped, or 0 if solid. The remaining bits will reflect the value of each ball, where the striped ball with the smallest number would be 1001 and the solid ball with the largest number would be 0111. This enumeration holds two separate logical statements in one integer, each of which can be used in further computation. For example, one computation can record the value of a ball when it enters one of the six pockets and determine whether the ball is striped or solid. Furthermore, the pockets can be numbered from 1 to 6 with 0 meaning no ball went into any pocket (for a prime state structure). Together, these Alethic subspaces can describe a game of billiards.

In the previous example, I could bind (enumerate) each ball to a set of attributes in a variety of different ways. Each enumeration has benefits and disadvantages with respect to the requirements of the Alethic space. The limitations of an Alethic subspace can be exploited and various binding algorithms can be applied to the same set of objects. In case where the width is 1, each ball can be identified and there is room for extra information or predicates. Whereas with width 4, each ball can be identified, and by using a particularly clever binding algorithm, the type of each ball (striped or solid) can be determined. As you can see, both enumerations can identify the set of balls, but they differ in the type of information stored by each binding algorithm.

## 14: Open Extensions over an Alethic Space

Up to this point, I have addressed finite globally closed Alethic spaces. We now turn our attention to open extensions of locally closed Alethic spaces. In a global closure, the underlying field contains all information about the Alethic space; whereas in a local closure, some additional information may be needed. An open extension contains that information, and these extensions are defined as the globalization of a local closure. One example of an open extension is the progression of dimensionality in an n-sphere. This progression relies on the addition of a point outside of the vector space that defines each successive sphere. At a local scale (i.e. the dimensionality is restricted to a specific value), the vector space is self-contained, and any sphere of the given dimensionality can be constructed. By adding a point outside of the vector space in which the sphere is embedded, I can complete the construction of all multi-dimensional spheres. Furthermore, I can define an infinite-dimensional sphere as one that is embedded in a vector space of all dimensions (i.e. the vector space expands to accommodate spheres of any dimensionality). In addition, a zero-dimensional sphere shall be defined as a single point in any vector space. This rings familiar as the concept of diminishing infinite spheres. Note that infinite spheres are infinite in radius not dimensionality, and infinite-dimensional spheres are infinite in dimensionality not radius.

The open extension of all spheres of any dimensionality and any radius can be realized as the Alethic space with a binding algorithm assigning to each sphere its dimensionality and radius. The logical search algorithm shall be defined by composing operators over the enumerations of each sphere. Also recall that these spheres can be defined by points that have some displacement from its surface; furthermore, these displacements define a vector in the dimensionality of the space in which the sphere is embedded (i.e. a 1-sphere is the boundary of a circle, and is embedded in a 2-dimensional vector space. In addition, the point that is chosen to be the origin does not add to the dimensionality, since the origin is merely a reference point. Therefore, the vector defined by the displacements is of the same dimensionality as the embedding vector space). Note that the origin need not be the zero point. The equations for a sphere can be rewritten in the following manner to define a particular transformation of a specified vector:

Let Mij be the cell at the ith row and the jth column of the matrix M, Aj be the jth component of the vector defined by the displacements, Cj be the jth component of the center, and Oj be the jth component of the origin. The points that must be used to construct the N-sphere are determined by the matrix M, and the displacements are determined by the vector A. Let ωij be the jth component of the ith point of the N-sphere and ∆i be the displacement of the ith point where ∆0 is the displacement of the origin. The following equations relate ω to M, and ∆ to A and ω:

ωij – Oi = ½ M-1ij ∆i – ∆0 = Σi (ωij2 – Oi2) – Ai

Finally, C=MA and A=M-1C as matrix products; furthermore, an Alethic product can be determined by AT=CTMT and CT=ATMT, where AT is the transpose of A and MT is the inverse of the transpose of M, and AT is an algebraic vector, CT is a logical vector, and M is the conversion tensor.

Any Alethic space can be represented by a multi-dimensional sphere. Take for example the game of billiards from the previous chapter. Let the dimensionality of a set of spheres correspond to the width of the enumeration of the balls, and the displacements be the values of each ball. An obvious operator over this space is the identity wherein each sphere is mapped to itself. Another operator could be defined by the number of balls that remain for either player. Furthermore, I can define any transformation over the set of spheres by mapping their displacements to their centers with a set of points corresponding to the transformation matrix. In this way, a more sophisticated game of billiards can be constructed.

Note that there are two distinct classes of spheres: those that pass through the origin (zero point), and those that do not. The spheres that pass through the origin have a zero displacement on the zero point. This class of spheres describes the operators over the set of balls. Note that calling the origin a zero point is a bit misleading, since the components of the origin need not all be zero. However, the origin acts as an initial point of the vectors defining the N-sphere. Consequently, I shall call it the zero point (the length of a vector from the origin to the origin is zero).

The second class of spheres shall describe the balls themselves as the displacement of the origin from the surface of the corresponding sphere. The spheres that do not pass through the origin can be defined by any set of points and displacements; however, the matrix representation of each sphere is determined exclusively from the points chosen. Furthermore, the displacements correspond to the elements of a vector to be transformed. For example, an operator can affect a specific ball, and the displacement of the origin from the surface of the sphere can, as described above, represent that ball.

Recall that each sphere can be determined by the product of a matrix and a vector (cf. an Alethic product). The notion of representing an Alethic space by a multi-dimensional sphere builds from this correspondence. Furthermore, the unique point of intersection of a set of hyper-planes can be determined by the center of a single sphere. This is particularly interesting since the sphere will be orthogonal to each of the hyper-planes (the center of the sphere is the intersection of the hyper-planes; each of which cut the sphere radially, thereby intersecting the sphere at right angles). It is convenient that both structures can be described by matrices, since they can thusly be mapped onto each other. Furthermore, an Alethic space can be determined by either a set of multi-dimensional spheres or a set of hyper-planes.

Another type of open extension consists of a series of derivatives of a function (cf. operator). If a set of k points on either the function or its derivatives is known, the function and its first k-1 derivatives can be determined at any point. However, this series converges only if the derivatives of the function approach a linear progression (i.e. the kth derivative Dk is less than kDk-1 for large k, where D is the differential operator). The following are forms of the series:

⊕r is an ordered tuple whose elements are of the form ri, whatever r may be. The first series represents the value of a function of a single variable at one point, evaluated at another. The next equation shows a change of basis in a function of many variables. The final equation generalizes the second to the tith derivative of the function with respect to the ith variable. The next equations are special cases of the last equation:

The first of these equations describes the affect of multiplying each variable of a function by a scalar, and the second shows the result of adding a scalar to each variable. Note that these scalars need not all be the same. These equations should remind you of a Taylor’s series, since the underlying principle is the same: find the value of a function at a point using a polynomial. In fact, I derived these equations by studying the behavior of a difference in the value of a polynomial evaluated at two separate points. However, I shall not divulge the details of the derivation here. What is important of these equations is that they can be used to describe a function when only certain points are known.

To this end, I could determine a function that has given values at specific points; furthermore, I can add extra information in the derivatives of that function. To include this extra information, I must extend the length of the polynomial representation of the logical structure. The following algorithm derives the polynomial that obeys a set of known conditions:

First, order the conditions lexicographically by derivative then by parameter (i.e. the conditions in the left column are ordered as in the right column):

1) f(1,0)(0,1) = 3 1’) f(0,0)(1,0) = 0

2) f(0,1)(1,0) = π 2’) f(0,1)(1,0) = π

3) f(1,0)(0,0) = Φ 3’) f(1,0)(0,0) = Φ

4) f(0,0)(1,0) = 0 4’) f(1,0)(0,1) = 3

Next, write the equations for each condition (i.e. we are looking for a general form of f as evaluated at the indeterminates u and v):

Then, we must normalize the equations (i.e. rewrite each equation to have the same starting index):

Observing that all terms are linear in derivatives of f, set up a system of linear equations:

Since there are four conditions, I must use a system of four unknowns. However, you should notice that there are three unknowns and one error correcting function (superscript denotes differential offset, and subscript denotes point) for each equation. Therefore, we must pull an unknown from the error correcting function. This extra term must have the same differential order for each equation; furthermore, the minimal such term must have as its differential order the maximal differential order as prescribed by the conditions. In this case, the minimal order is (1,1). If that order is already used, any term of greater differential order (lexicographically) may be included:

Now, I can build a matrix equation to represent the system:

Finally, I can solve for the vector of unknowns:

A similar algorithm applies for higher order conditions; just find a system of equations in the derivatives of f, and solve the matrix equation. Note that an inverse of the coefficient matrix must be used, and the efficiency of the determinant algorithm for matrix inversion has the order of k(k!). Thus, this is very cumbersome for a large number of conditions. Fortunately, this matrix equation can be represented geometrically as an N-sphere. Recall the following equations for the points of the N-sphere, and their displacements:

ωij – Oi = ½ M-1ij ∆i – ∆0 = Σi (ωij2 – Oi2) – Ai

Dually, the equations can be set up in terms of logic vectors:

ωij – Oi = ½ Mij ∆i – ∆0 = Σi (ωij2 – Oi2) – Li

Let M be the matrix of coefficients, and L be the logical vector of constants. If the origin is taken to be the zero point, and to have a displacement of 0, it follows that the points and their displacements may be the following (with u,v = 2):

Finally, we can simplify the displacements:

The function and its derivatives will be the components of the center of the N-sphere defined by these points and displacements. Notice that it was not necessary to find the inverse of the coefficient matrix to arrive at a meaningful geometric structure. This is one of the many advantages of using the duality of logic and algebra in the manner of Alethic Data.

To further this point, I can choose u and v to be any other elements in the open extension outside of the local closure of the conditions. The requirement that u and v to be values not used in the conditions is inherent of the principle feature of an open extension to actually be open (otherwise, singularity will occur).

## 15: Continuity and Limits on Open Extensions

For our next topic, we shall consider limits and the continuity of open extensions. By the standard definition, a limit at (⊕x) is the value that a function f(⊕u) approaches when (⊕u) approaches (⊕x) (i.e. if for any arbitrarily small neighborhood (⊕δ) about (⊕x) there is an interval on f containing L, then L is the limit of f at the limit point (⊕x)). Note that the value of f(⊕x) need not actually be equal to the value at the limit point. For example, f may not be defined at (⊕x), but all points adjacent to (⊕x) may approach the same limit point; thusly, f has a limit at the point (⊕x). The most practical choice of a neighborhood is a multi-dimensional sphere, one of which may be defined by the following:

Points: Displacements:

0, 0, 0, ..., 0 0

x1+δ1, x2 , x3,..., xn 0

x1, x2+δ2, x3,..., xn 0

x1, x2, x3+δ3,...,xn 0

... 0

x1, x2, x3, ..., xn+δn 0

The center of this N-sphere will approach (⊕x) when (⊕δ) approaches (⊕0), and if the values of the function at all points on the surface of the sphere each approach a common value L, then (⊕x) is a limit point of f with the value L. If the value of f at the boundary of the neighborhood of (⊕x) cannot be maximally contained, but a limit exists at (⊕x), then the limit of f at (⊕x) is ∞. Similarly, if the value of f at (⊕x) cannot be minimally contained, but has (⊕x) as a limit point, then the limit of f at (⊕x) is -∞. Continuity at a point (⊕x) can be defined by the following conditions:

1. The limit L exists at the point (⊕x) and is finite
2. The value of f is equal to L at (⊕x)

If all points in the domain of f satisfy the above conditions on f, then f is a continuous function. Continuity is an important feature of the real numbers, and can thusly be extended to complex numbers (i.e. complex numbers can be represented as a pair of real numbers; a real part, and an imaginary part). This property of a function of a set of real or complex variables allows for the construction of analytic structures such as derivatives and integrals. Another type of function can be defined discretely with variables in a ring isomorphic to either the integers (i.e. **Z**), multiples of a prime (i.e. p**Z**), or the integers modulo a prime (i.e. **Z**p). These three rings form a Euclidean domain with Euclidean function **Z**p+p**Z**=**Z**. Furthermore, **Z**p is isomorphic to the quotient ring **Z**/p**Z**, and since **Z**p is a field, p**Z** is a maximal ideal of **Z**. The Euclidean domain of integers allows other analytic structures to be constructed such as sequences and series. The key distinction between discrete functions and continuous functions is that the distance between adjacent elements of a discrete function does not vanish in the way of continuous functions (i.e. the only limit points of a discrete function are those with coordinates only at positive or negative infinity, whereas all points of a continuous function are limit points). Note that the limit of a function at a point with coordinates at positive or negative infinity may be finite (i.e. the limit of x/(x+1) as x approaches infinity is 1). Limits of discrete functions are defined as the limit of the sequence described by the function as its variables approach positive or negative infinity. I will address sequences and series in a later chapter.

## 16: Differentials, Infinitesimals, and Adjacency

An infinitesimal on a continuous interval shall be defined as an object with the limit of 0, where the limit is taken at infinity. The domain of an infinitesimal is irrelevant to their definition, as they describe only a vanishing element. The concept of an infinitesimal can be likened to the infinite singularity (i.e. a logical universe of one dimension). Recall that I first introduced this structure by taking a sphere with infinite radius and attempting to find its center. There, I showed that not only does this point not exist uniquely, but that it could contain the entire space of an infinite sphere. Of course, these two definitions seem to contradict each other, but recall that the center could be anywhere within the infinite sphere, and further, could it not be everywhere? The infinite singularity is a device to address structures that yield contradictions, but must be in some way measured.

On the other hand, differentials are infinitesimals with context (i.e. directions or some other distinguishing attribute). A differential of an algebraic structure has context in the form of instantaneous change. However, a logical differential could have any context that describes the flux of causality (i.e. which events influence others and how does that change over some interval). Note that the aspect of an interval over which a logical structure varies is purposely left ambiguous as to allow any suitable structure be used. In either definition (algebraic or logical), differentials describe instantaneous change. Furthermore, partial derivatives can be realized as the ratio of differentials affecting only one dimension at a time; just as they are defined for functions in real variables. In Alethic Data, differentials are used in continuous functions to relate adjacent points or events. Logically, any two events are adjacent if there is a causal link between them; alternatively, two points are algebraically adjacent if they differ by only an infinitesimal. Take for example the game of billiards, as before. Two events that are adjacent are the instant the cue ball is displaced and the instant the cue ball collides with any of the other balls. The act of displacing the cue ball provides the possibility of a collision at a later instant. Another set of adjacent events are the sinking of a ball and the next turn: if the player sinks one of their balls without “scratching” the cue ball or any of the other player’s balls, then they have the next turn; otherwise, the other player takes the next turn. These two different notions of adjacency can be combined as no ball can be sunk successfully without a collision. The flux of causality can here be defined by collisions and turns: a collision allows a ball to be sunk which may then add to the player’s turn or progress the turn to the next player.

Logical adjacency is an important feature of Alethic Data, and the flux of causality can be used to distinguish and relate “consecutive” events. In an algebraic environment, adjacency can be defined as the behavior of a function in the vicinity of a set of points (i.e. a locally closed Alethic space). For example, I can take all 2-spheres centered at a single point as a topological basis of Euclidean 3-space. Adjacency would then be defined by the diminishing distance between 2-spheres (recall that two points are adjacent if they differ only by a infinitesimal). Logically, adjacency can be described by a set of events that happen sequentially or simultaneously. For example, the moment that the cue ball is struck is the same instant that the cue ball begins to move. Alternatively, a collision can occur after the cue ball is struck, thus these two events are also adjacent.

Recall that an Alethic operator must be predictable, reliable, and have either duality or closure. It is to be shown that a logical differential is an Alethic operator. To show predictability, observe that a differential describes the change of a function as one of its parameters changes. The set of functions that are used to describe the logical structure as it arises can be observed differentially when very small deviations in parameters are studied. To this end, the differentials can be predicted by those functions. Each time the functions are run over identical parameters, they must return exactly the same result. In this way, logical differentials are reliable. Finally, duality can be realized through the sum of changes in a function over many parameters (i.e. integration is the dual of differentiation); furthermore, logical differentials are dual to algebraic differentials as there is a projection between logic and algebra. Thusly, logical differentials are Alethic operators.

The structure of an Alethic operator for an open extension has a similar form as in a global closure. Adjacent elements of a global closure have integer differences; alternatively, adjacent elements of an open extension have infinitesimal differences. This distinction allows an open extension to be a more general form of a global closure. As was introduced in the section on open extensions, when a finite set of conditions is known, a general form of a representative function can be constructed. Furthermore, the first few derivatives of the function can be determined simultaneously. This provides a useful parameterization of different events and allows many operations to be performed simultaneously by using a single polynomial. In addition, these operators can be realized through multi-dimensional spheres (i.e. the center of an N-sphere can represent the action of an operator). Recall also that by using an N-sphere, I can eliminate the necessity for matrix inversion, since I can geometrically describe the action of solving a system of linear equations by assigning the center to be the value of an Alethic product. Finally, these objects (the centers of a set of N-spheres) form an Alethic vector field, as will be addressed in the next chapter.

## 17: Alethic Vector Fields

An Alethic vector field is a set of objects that describe an event (such as the action of an operator), and such a field can be defined by a set of parameters over an Alethic space. An N-sphere is embedded in an Alethic space of N+1 dimensions, and vectors can be built from an origin point of that sphere. It should be duly noted that N+2 points are needed to define an N-sphere, and the center will be in N+1 dimensional space. Any of the N+2 points can be chosen as the origin of the sphere, and vectors can be constructed with initial points being located at the chosen origin and terminal points being each of the other N+1 points. If the origin is taken to be the zero point, and is located on the surface of the sphere, then the Alethic vector field of the sphere is reduced. This type of sphere is most useful because it simplifies the calculations that determine its attributes. Furthermore, the radius of a reduced sphere is the norm of the center, and can therefore be used to store additional information about the data that the N-sphere represents. The norm of a vector should be thought of as a metric that describes its size.

Logical differentials form an Alethic vector field, since they have context and value (these being necessary and sufficient conditions to determine that a vector field is Alethic, cf. predictability and reliability). Algebraic vectors have magnitude and direction (cf. value and context, respectively), and thusly, they form a special type of Alethic vector field (AVF). More generally, an AVF can be written using any suitable coordinates. For example, the coordinates of an N-sphere can either be the set of points and displacements used to define its structure, or they could be its center and radius. Either way, additional information can be stored in these coordinates. Furthermore, coordinates can be chosen to optimize the structure of some algebraic or logical event. An example of an optimal coordinate system can be determined as before by the following equations:

ωij – Oi = ½ M-1ij ∆i – ∆0 = Σi (ωij2 – Oi2) – Ai

ωij – Oi = ½ Mij ∆i – ∆0 = Σi (ωij2 – Oi2) – Li

Where ωij is the jth coordinate of the ith point, and ∆i is the displacement of the ith point from the surface of the sphere. The top equation can be used if a logical structure is to be determined, and the bottom equation should be used to determine an algebraic structure. In addition, depending on the direction of a transformation, either could be used to optimize inversion. From these equations, I can determine the points used to define an N-sphere, and their displacements. From the infinite singularity and open extensions over an Alethic space, I can determine the center and radius of the N-sphere representing a particular set of conditions. These statements should appear very similar to each other, since they describe the same structure. However, it is important to observe that these are not identical, since the properties of either set of parameters are very different. The first of these statements is used when conditions are known and it remains that the data is to be represented geometrically; the other is used when an Alethic product is to be performed (the result is the center of the sphere).

Recall that an infinite-dimensional sphere is a geometric device to generalize the notion of a spherical locus. This type of sphere can be defined not only by a sphere of all dimensionalities, but as an inductive surface. This latter definition is useful when there is a set of parameters that are to be combined and analyzed simultaneously. For example, suppose I wanted to build a game that consists of a matrix of cells where each cell specifies the direction from which a player can leave that cell (let the matrix represent a labyrinth). Let there be a chance of encountering an enemy at any cell, and the encounter leads to a battle. In the battle state, there shall be three events that happen cyclically (the enemy’s attack, a menu for the player’s turn, and the response to the player’s choice from the menu). The battle can start at either the first or second events, but the third event must be preceded by the second, and the events happen sequentially after the initial event has occurred. I can represent these three events on the same sphere by using a block diagonal matrix of any number of events; furthermore, I can leave the number of events indeterminate. Since the initial event is unknown until it occurs, an infinite-dimensional sphere can represent the entire battle sequence geometrically without the process of the battle ever running. When the battle sequence runs, a specific state of the infinite-dimensional sphere is determined, and this sphere shall now be finite and representable by a matrix and a vector (or a center and a radius). An infinite-dimensional sphere can be thought of as a movie, and a specific state would then be a single frame. The construct of an infinite-dimensional sphere can be used when many events can happen, but it is not known when they shall occur until their instantiation. In this way, I can model a sequence of events in a predictable and reliable manner without actually running the processes that describe them.

Recall that if the origin is the zero point and is on the surface of the sphere (i.e. the sphere is reduced), then the radius represents the norm of its center. By this, I mean that the norm of a vector initial at the origin and terminal at the center is the same as the radius of the sphere. It should be noted that the center of a sphere could be defined as the result of an Alethic product; therefore, the center of an infinite-dimensional sphere shall be defined as the algebraic map of a sequence of adjacent events, or the logical equivalent. If the origin is the zero point and is on the surface of the infinite-dimensional sphere, then the radius of that sphere is a metric that describes the act of performing the Alethic product. Furthermore, once an infinite-dimensional sphere is determined (i.e. has been assigned a radius and center) it can be assigned a vector space, using the origin as the initial point and any set of linearly independent (no N points lie on a hyper-plane in N-2 dimensions or less) points at some displacement from the surface of the sphere as the terminal points to form the basis vectors of the space.

## 18: Sequences and Series

A sequence of events can be denoted by the algebraic structure of a sequence of logical events, and a series would then be sum of all elements of such a sequence (i.e. the metric of logical causality). The terms of an infinite sequence can be represented as the squares of the coordinates of the center of a reduced (origin is the zero point and is on the surface of the sphere) infinite-dimensional sphere, and the square of the radius would then be the sum of the elements of the sequence. Note that these definitions give rise to a sphere of any number of dimensions (just terminate the sequence at a specific term that corresponds to the dimensionality of the sphere). By using a sequence to define a sphere (cf. Alethic product), I can represent a set of adjacent events each of which would be the square of one of the coordinates of the center of the sphere. The position of the event as a coordinate of the center can be determined as its position in logical causality. For example, the event could be a battle from the labyrinth game, and the center could be of 3 dimensions (the stages of a battle are three-fold and cyclic) where each coordinate represents the value of each stage of the battle. One example of such an enumeration could be the damage inflicted by an attack (in an attack stage) or the choice of the player when the middle stage (i.e. the choice stage) is encountered. The function to build the sphere could be called multiple times to simulate the entire battle sequence. If the correspondence between the values of each stage and their coordinate in the center is chosen cleverly, I can use the radius to model the total damage inflicted as well as record the choice of the player. For example, I can choose the mapping of the damage from the enemy’s attack to be denoted by an integer with length 2 in base 16 (minimum being 0, maximum being 255), the choice can be an integer of length 2 in base 4 (allow for 16 different options), and the damage inflicted by the player could be denoted by an integer of length 2 and base 16 (same as enemy attack). These values can then be weighted by their locations (the enemy’s attack would have weight 1, the choice would have weight 162, and the player’s attack would have weight 163). When the values of each stage have been calculated, the square of the coordinates of the center can be denoted by the values of the weighted stages, and finally, the square of the radius would be an enumeration that encodes all three stages of the event of a battle.

The weights above are chosen in order to separate each stage and allow them to be combined without interfering with each other. The first weight shall always be 1, since there are no preceding stages. The second weight shall be 1 more than the maximum state of the first stage. Subsequent weights shall be the product of the weight of the previous stage and one more than the maximum state of the previous stage. In this way, the weights may be calculated inductively; furthermore, an infinite-dimensional sphere can be defined by the sequence of adjacent events (stages) performed during the battle (i.e. each coordinate can be inductively chosen to model each stage of the battle sequence). To pull information from the square of the radius (each element of the center must have the same underlying base; this is satisfied in the above example in base 16):

1. Divide the metric denoted by the square of the radius (i.e. the sum of all stages) by the weight of the element to be computed (163, 162, 1 for a three dimensional sphere [1 cycle] and 168, 167, 165, 163, 162, 1 for a six dimensional sphere [2 cycles], and so on).
2. Then, take the integer part of the quotient and find the remainder modulo the base of operation.

This algorithm pulls a specific stage out of the radius of a sphere, and it can be repeated to examine any subset of information encoded in the radius. It is convenient that the radius and center of a sphere can be calculated from each other using weighted stages. Furthermore, it follows that particular points in N+1 space and their displacements can be calculated. In this way (using non-weighted coordinates), the Alethic product can be reversed. This is particularly useful when it is required that many conversions be performed. Note that Data spheres (i.e. data is stored in the radius of the sphere by taking events as terms of a sequence) are distinct from Alethic spheres (i.e. data is stored in the center of the sphere by an Alethic product), but they can be used together to represent the same event. Hence, the title of this project is Alethic Data (referring to both types of spheres being used simultaneously).

Recall that the components of the center can represent terms of a sequence, and the radius can follow as the sum of each term of that sequence. This gives rise to a third type of sphere distinct from the other two (a Σ-sphere). The following equations define the coordinates of the center, Ci, and radius, R, of a reduced Σ-sphere:

Ci2 = Ai R2 = Σ A

Another type of sphere can be defined by a system of linear equations (a λ-sphere); furthermore, this sphere will intersect each hyper-plane of the equations orthogonally. The following are equations that define the points, ωij, and their displacements, Δi, for a reduced λ-sphere:

ωij = ½ M-1ij Δi = Σω2 – Ai

Let us take Σ-spheres and analyze them more closely. This type of sphere has information encoded in its center and radius in the form of a sequence and a series, respectively. If the center is to be analyzed, an infinite-dimensional sphere will have as a limit the value of the sequence as the index approaches infinity. However, if the radius is to be analyzed, an infinite-dimensional sphere will have as a limit the value of a series whose terms are that of the sequence defined by the center. It is important to note that these values are encoded as the squares of the center coordinates or the square of the radius.

Now a sequence of adjacent events can be defined by an Alethic Data pair. The Alethic sphere shall be used to denote the transformation of a logical structure to an algebraic one or vice versa, and a Data sphere shall encode the values of a sequence of events. When events that are adjacent are analyzed, the distinguishing metrics are differentials, and the radius will approach the area under a curve connecting each event. This curve can be encoded as the set of logical values interpreted algebraically. The following chapter shall construct a definition of logical integration (think of the effect of integration on a real-valued function).

## 19: Integration on Extension Spaces

As described in the previous chapter, a sequence can be encoded into the square of the coordinates of the center of a Data sphere, and the square of the radius of this type of sphere is the value of the representative series. When all terms denote adjacent events, then the distinguishing metric of these events is a differential. By incorporating this component as a causal deviation (a metric that represents the progression from one event to the next), I can define integration over an extension space. To this end, allow there to be definitive initial and terminal events such that all events that occur between those events are adjacent. If this is the case, the series defined by the Data sphere can be represented by an integral. This concept follows from the typical formulation of the fundamental theorem of calculus:

Which can be rewritten as:

The left side of the equation represents the metric described by performing integration over a set of adjacent events, and the right side denotes the value of the metric. Note that the left side is a difference of the values of a function at two distinct points, and the right side is a sum of derivatives. If all events are adjacent, then the fundamental theorem of calculus can be reduced to:

Since multiplication in these equations is commutative and associative, I can conclude the following statement in one variable:

This should look familiar to anyone who has studied calculus, since it is the first form of the fundamental theorem of calculus that you will encounter (usually prime notation would be used in place of the differential operator Dt).

Integration as traditionally defined is the sum of differential elements in the region of a bounded algebraic curve. The fundamental theorem of calculus shows that the integral of the derivative of a function is the function itself, and when evaluated, constitutes a difference in the values of that function along the boundary. This subtle result is critical in the logical definition of integration. As the frequency of measurement increases, so does the resolution of the result. So logically, an integral shall be defined as the sum of the representative regions of the results of a system under continuous data collection, ergo infinite resolution. However, as new data points are introduced, the complexity of the system increases. For this reason, integration can be used to model a system with infinite resolution obeying some set of conditions. The integral of a logical structure can therefore be defined as the limit of the anti-differential metric of a system as the resolution approaches infinity (i.e. as the intervals between observations decrease, the resolution and therefore the accuracy increases). It would be impractical to take an infinite number of conditions; however, any finite collection of conditions may be employed to isolate a curve with specific properties. Examples of a finitely generated collection could include a finite number of conditions that depend on a variable expression (all conditions that satisfy the expression must be included in the collection, and there may be infinitely many such conditions; however, they may be finitely generated).

## 20: Non-Cartesian Coordinates

In this concluding chapter, I shall endeavor to define non-Cartesian coordinate systems to simplify the construction of the algebraic representation of a logical event. From calculus you should be familiar with spherical, polar, and cylindrical coordinates. From intermediate algebra, you should be familiar with the complex plane. Also, you should be familiar with parametric equations. Recall that a system of linear equations can be represented by a multi-dimensional sphere. This sphere has as its center the intersection of the hyper-planes described by those equations, and is normal at its intersections with each of the hyper-planes. These two properties together allow a change in basis from linear components to spherical components. If all hyper-planes intersect each other at right angles, an orthonormal basis can be defined by unit vectors orthogonal to each hyper-plane individually. Coordinates can then be the components of a linear combination of the unit vectors (i.e. in 3-dimensional Euclidean space, Cartesian coordinates of a point can be defined as the number of unit vectors in each direction required to describe that point). Alternatively, spherical coordinates can be used to define the same point in an N+1 dimensional space, but here, there shall be one linear component, and N angular components. The linear component describes the distance a point is from the center of an N-sphere, and the angular components describe the angle that the projection of the vector from the center of the sphere to that point onto a space of lesser dimensionality. The following equations give parametric equations for a 3-sphere (in 4 dimensional space):

Spherical to Cartesian:

Cartesian to spherical,

These equations illustrate how a point can be denoted in two different coordinate systems. It should be duly noted that these two systems model the two distinct ways to represent a sphere (by a set of points in a rectangular system or by a center and a radius in a spherical system). Furthermore, these systems can be extended to N dimensions.

Spherical to Cartesian:

Cartesian to spherical:

However, these forms are not ideal for spheres whose centers are not the origin. Instead the following should be used, where (a0, ..., ai, ..., aN-1) is the center of the sphere:

Spherical to Cartesian:

Cartesian to spherical:

These forms allow one to describe spheres with any center; furthermore, a sphere that is defined by a set of points and displacements can in this way be determined (as by center and radius). Therefore, an Alethic Data pair or any other spherical data representation (cf. λ-sphere, Σ-sphere, etc.) can be determined under either spherical or Cartesian coordinates.

In conclusion, there may be many ways to represent the same set of data as the need arises. I shall leave it to the reader to explore the various applications of Alethic Data. The underlying purpose of this work is to demonstrate the formulation of a duality of logic and algebra. To begin you may observe the behavior of an Alethic Data pair over a set of events (i.e. compare the action of conversion with the action of composition).

There it is, Alethic Data.

# Appendix A : The Labyrinth Game

In this section, I shall lay the groundwork to develop a game by employing techniques from Alethic Data. First of all, I must decide what kind of game I want to build. Let us take as our example a labyrinth game. The object of this game would be to progress through each level by either finding the exit, or by defeating enemies and gaining experience. It is critical to understand how the game shall be structured, both logically and algebraically. There are several parts of this structure and I shall address them systematically.

The first part I shall discuss is the structure of the board in which the player navigates. The board shall be defined by a square matrix of hexadecimal characters that denote the directions from which the player can leave a cell. Let up be represented by 8 (i.e. 1000), right be 4 (i.e. 0100), left be 2 (i.e. 0010), and down be 1 (i.e. 0001). Furthermore, if the player cannot leave a cell, the player loses and is teleported to the starting point of the board. Combinations of directions can be represented by a sum of representative integers. The location of the player shall be represented by a row and column pair, and the value of the cell corresponding to that pair shall describe the directions that the player can move. Two cases follow; success and failure. In the successful case, the location of the player is updated. In the case of failure, the location of the player remains the same and an error message may be displayed stating that the choice was not a valid direction. After the player’s move has been validated as either case, there is a chance of an enemy encounter.

If an enemy was encountered, then a battle begins between the player and the enemy. There are two possible starting states: the enemy has the first turn, or the player has the first turn. Each enemy and the player shall have specific attributes, and these will be denoted as integers. Based on the values of each attribute for the enemy and the player, some amount of damage may be inflicted. When the enemy attacks, the attack power of the enemy is randomly selected with its upper bound being the attack statistic of the enemy, and the defense resilience of the player is similarly selected from the player’s defense statistic. The damage inflicted is the difference of the defense resilience from the attack power. If this number is not positive, then player has evaded the attack. A similar procedure applies for the player’s attack against the enemy (just replace enemy with player and vice versa). However, the player’s turn shall always begin with a menu that gives the player a variety choices by which they may proceed. For example, there may be choices to attack, run, or cast a spell. The battle phase terminates when either the player or enemy is exhausted of health, or the player successfully runs from the battle.

Next we can add items that can be picked up at various locations on the board. Similar to the directions matrix, a pickup matrix may be used to assign specific cells with an item. Let us suppose that there are 6 types of items: missiles, bombs, teleporters, maps, health potions, and mana potions (to cast spells). Missiles can be fired in any of the four directions. When the missile is fired, all walls in the direction chosen (horizontal or vertical walls; perpendicular to the direction in which was chosen) until either the edge of the board or a zero cell is encountered. Bombs shall eliminate all walls within a one cell radius, zero cells are also affected (cf. missiles). Teleporters can send the player to any cell in the board matrix of the level. Maps display the board as a user friendly diagram. Finally, health and mana potions replenish some of the player’s health or mana, respectively.

Alternative to choosing a direction, the player may open an inventory and option menu. To open this menu, the player can type the inventory command (type i, for example). This menu shall contain information about the player’s items (those that can be picked up). In addition, options shall be specified to select an item. Other options could be used to do other functions, such as display the player’s statistics, exit the inventory menu, quit the game, or cast a spell. When the option chosen is to quit the game, there shall be a choice to save the game. If instead the selection is to cast a spell, a new menu shall be displayed containing the spells that can be cast. These spells may be different between battle and non-battle phases.

As for missiles and bombs, they can be represented by Alethic operators on the cells of the board matrix. The value of the cells of the missile or bomb operator shall reflect the directions in the board matrix that are opened by either weapon. The following matrix shows a labyrinth and each cell has a number corresponding to the directions in which the player can leave that cell:

|  |  |  |  |
| --- | --- | --- | --- |
| 4 | 3 | 5 | 2 |
| 1 | C | B | 1 |
| D | 6 | B | 9 |
| 8 | 0 | E | A |

The cells of this matrix have indices starting with (0,0) at the top left corner and ending with (3,3) at the bottom right corner. If the player is at location (2,1), then the player can only move left or right (0010 + 0100 = 0110 = 6). The boundary between cells (3,1) and (3,2) denotes that the player may move leftward but not rightward across the boundary (the player can enter the cell on thin side of boundary, but cannot exit from the cell on the thick side of the boundary). The following diagrams illustrate the effects of a weapon on the initial board. The left matrix shows the values of the resulting board layout, and the right matrix can be bound to the initial matrix by the logical operator “or” (algebraically, P + Q + PQ). In other words, or is a mapping that combines the initial matrix with the matrix on the right to build the resulting left matrix.

The next two matrices simulate firing a missile down from location (1,1):

|  |  |  |  |
| --- | --- | --- | --- |
| 4 | 3 | 5 | 2 |
| 1 | D | B | 1 |
| D | F | B | 9 |
| 8 | 0 | E | A |

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 9 | 0 | 0 |
| 0 | 0 | 0 | 0 |

And the following matrices simulate firing missile leftward from location (3,3):

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 6 | 2 |

|  |  |  |  |
| --- | --- | --- | --- |
| 4 | 3 | 5 | 2 |
| 1 | C | B | 1 |
| D | 6 | B | 9 |
| 8 | 0 | E | A |

Note that this matrix is the same as the initial matrix since the trap absorbs the missile (i.e. the boundaries between cells (3,0) and (3,1), and cells (3,1) and (3,2) are unaffected). The following matrices show the effects of deploying a bomb at the cells (2,1) and (3,2), respectively:

|  |  |  |  |
| --- | --- | --- | --- |
| 4 | 3 | 5 | 2 |
| 5 | F | B | 1 |
| D | F | B | 9 |
| C | E | E | A |

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 0 | 0 | 0 |
| 5 | 7 | 3 | 0 |
| D | F | B | 0 |
| C | E | A | 0 |

|  |  |  |  |
| --- | --- | --- | --- |
| 4 | 3 | 5 | 2 |
| 1 | C | B | 1 |
| D | 7 | F | B |
| 8 | C | E | A |

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 5 | 7 | 3 |
| 0 | C | E | A |

Notice that the zero cell is affected, and that the extent of a bomb is the interior of the region whose boundaries are the cells adjacent to the player’s location. Each interior wall is broken, even if one of those cells is a trap. Other weapons can be introduced as the programmer desires. For example, a laser can be defined like a missile, but can shoot through traps.

Now that the basic parameters of the game are known, a structure can be introduced to compact the entire board into a single function. To do this, build an open extension over the labyrinth. In each of the labyrinths above, there are 16 cells; each cell shall be designated a value of a two dimensional function. From the result of chapter 14, I can construct such a function. The conditions used to build the open extension shall depend on the row and column of the cell (i.e. f( row, column ) shall equal the value of the corresponding cell of the labyrinth, where f is the function to be determined).

That is pretty much all that is required to build the Labyrinth game. However, other attributes can be added such as boss encounters, multiplayer mode with or without PvP collisions (like enemy battles but with another player), equipment that can be worn or used by the player, and whatever else the programmer can think of.

# Appendix B: Geometric Influences on the Alethic Involution

Recall that an involution is a structure the is its own inverse. The Alethic involution, AI, is described by 2-state logic and Boolean algebra, since in this unique case, the conversion tensor from algebra to logic is the same as the inverse conversion tensor from logic to algebra. This is also the structure of the standard logical universe, which contains three domains: statements, predicates, and operators. From chapters 9 and 10, if a two-state self-dual statement is divided into two halves of equal length, the left half will also be self-dual and the XOR sum of the left and right halves will be the dual of the right half. The following theorems describe the closure of the XOR sum over a pair of self-dual statements.

(TB.1) The XOR sum of two self-dual statements in the AI is self-dual.

Proof: Let A and B be self-dual statements in the AI. By the definition of self-duality, A=AM and B=BM. Let φ(A,B) be the XOR sum of statements A and B. φ(A,B)= φ(AM,BM) =AM + BM = (A+B)M = φ(A,B)M. Therefore, φ(A,B) is self-dual.

The transformations under the AI can be canonicalized by mapping through three-state Alethic geometry. This can be achieved by the following criteria regarding the factors of each term stated in the pre-image of an element of the Alethic vector field (AVF):

1. If the term contains the negation of a variable, the image of that variable of the that term is unity.
2. If a variable is absent from a non-zero term, the image of that variable of that term is its negation.
3. If a variable is present in a term, the image of that variable of that term is itself.

These transformations obey the following criteria:

1. Zero maps to zero
2. Products map to products
3. Sums map to sums

In this way, I can represent the image (valuation by coefficients) of the element 1+P(1+Q) (i.e. P implies Q) of the AI in three variables as (1+P)(1+Q)(1+R)+P(1+R) or 1+R+Q+RQ+QP+RQP. Furthermore, since negation and unity are geometric duals under these criteria (each maps to the other) and presence is dual in a similar way to itself (if a variable is contained in a term, it is also contained in the image of that term). Finally, terms that are entirely absent from an element of the AVF are absent (in the sense that they are not transformed) from the image of that element (zero maps to zero).

In another manner of evaluation, the information encoded in an Alethic Data pair can also be analyzed geometrically. The Alethic component of the pair encodes the conversion of a vector between types (i.e. Alethic product), and the data component describes a set of parameters over a vector space. In either case, the region that contains information is the locus of all points at a given radius from a single central point. Furthermore, this region can be optimized to yield specific data as required by a set of parameters. To this end, it is required only that an N-sphere be determined by either a set of N+2 points in N+1 dimensions with a displacement vector, or by a center and a radius. However, these spheres are specifically designed to be mathematical models of information and need not be real (i.e. the radius, and the displacements and coordinates of each point can be a negative number or even complex). This being the case, one may be inclined to define how such an object can be realized and geometrically interpreted. The basic concept of an Alethic Data pair is to provide a framework through which any collection of criteria may be satisfied. Three basic properties for a data sphere must hold for complex valued criteria.

The first involves a complex radius. If all points and displacements used to define the sphere are real, then the radius must be either entirely real or entirely imaginary. An imaginary radius indicates that the displacement of one or more points is greater than the distance from those points to the center of the N-sphere. Therefore, the square of the radius must be negative and real (the radius itself would then be imaginary). On the other hand, if any point has one or more components in the set **C**\**R**, then the radius may have both a real and an imaginary part. Similarly, if any of the displacements are in the set **C**\**R**, the radius may again have both a real and imaginary part. Furthermore, these properties of the radius may be used to determine whether an imaginary part is contained in any conditions that determine the sphere, and therefore, the Alethic space.

The second involves a set of points whose coordinates have a non-empty intersection with **C**\**R**. The definition of the mathematical device of a data sphere shall remain as the locus of points at some radius from a central point. The union of all scalars of the center and radius of such a sphere shall have a non-empty intersection with **C**\**R**.

The final property involves the set of displacements of each point from the complex surface of the data sphere. If any of the displacements are a member of **C**\**R**, then the union of all scalars of the center and radius, again, has a non-empty intersection with **C**\**R**.

Geometrically, these properties describe the effects of complex data on the mathematical space of a data sphere. These properties, along with the two sets of criteria of mapping in the Alethic involution as previously described, form a mathematical construction in complex N+1 dimensional space. As should be duly noted, data spheres defined by complex valued points/displacements do not qualify as two state logical or algebraic structures. However, some meaning can be constructed from complex coordinates: angularly, the units of measure shall be the deviation of a point from the polar axis; radially, the units of measure shall be the deviation of a point from the polar center (cf. Alethic Data pair represented in spherical coordinates).

# Appendix C: Polynomial Reduction

A *monomial* *of* *degree* *d* is an algebraic structure that consists of a product of *d* not necessarily unique variables xi and a scalar *k*. If all *d* variables are identical, the monomial is of the form *k*x*d*; in general, a monomial is of the form *k*Πxij, where Π denotes a product in i and j with the sum of all j being *d*, i is the index of a specific variable, and j is the multiplicity of that variable. A *polynomial* is a sum of monomials of any degree. A *term* of a polynomial is defined as a monomial element of that polynomial. The *initial terms* of a polynomial are those with maximal degree within that polynomial. The *degree* of a polynomial is equal to that of its initial terms. A term is *present* in a polynomial if its coefficient is nonzero. A *monic* polynomial is one in which all present initial terms have coefficient 1. A polynomial in which all terms are of the same degree shall be called *homogeneous*.

(TC.1) The product of any number of homogeneous polynomials is homogeneous.

Proof: all terms of any factor have identical degree and the degrees of the terms of the product of two factors are the sum of the degrees of the factors. By induction on the number of factors, the theorem follows.

Monic homogeneous polynomials are of particular interest, since each present term has coefficient 1, and the degree of all terms are the same. Furthermore, a *leading term* shall be defined as the term with greatest lexicographic order (i.e. partial order over the multiplicity of each variable by index). In addition, the sum of a system of all monic homogeneous polynomials in *n* variables can be used to describe any two state Alethic space in *n* dimensions. When represented logically, the multiplicity of each variable of a term denotes the value that shall be substituted for that variable, and its coefficient shall be the value of the algebraic form when the specified substitution is performed. The algebraic form of the logical representative polynomial can be determined by applying an Alethic product on the coefficients of such a lexicographically ordered polynomial. Similarly, the sum of any system of homogeneous polynomials in *n* variables can be used to represent any Alethic space of *n* dimensions. Such a system shall be called a *homogeneous system of polynomials*.

Recall that a self-dual member of the Alethic involution (AI) contains the sum of its lexicographically lower half and the dual thereof in its lexicographically upper half. Furthermore, these halves can be logically separated by the multiplicity of the variable of highest index. This relation is unique to two state structures, and it is convenient that modern computing architecture uses such structures. However, think of the two state restriction as anything that can be represented in a binary based numbering systems. Unfortunately, statements in a binary numbering system are much longer than the representative structure in any other positive integer based numbering system. Yet, the advantages of using a binary numbering system include lexicographic self-duality. This feature of binary numbering systems can be used to combine logical data with its algebraic form to build a single structure.

I shall conclude with the particularly profound insight of deterministic polynomials, which are logical structures that map all elements of an algebraic polynomial simultaneously. There are a set of criteria that define a deterministic polynomial:

1) The product of a set of single-variable algebraic polynomials such that the variable is unique to each factor can be directly translated into a deterministic polynomial in some prime modulus.

a) To transform each factor, build the single-statement conversion tensor in the specified modulus,

b) Then, multiply each row by the coefficient of the term with the corresponding exponent (i.e. if the modulus is taken to be 5, then the effect of the term 2x3 is to multiply row 3 of M5 by 2) [row index begins at 0].

c) Next, add together by columns, the resulting rows; the deterministic polynomial in that variable shall have as its coefficients the elements of the row vector of the final sum.

d) Continue for each variable, noting that variables not present in a product of polynomials that are elsewhere used in the function are algebraically unity; therefore, they are transformed as a deterministic polynomial with all coefficients being 1 (row 0 be exactly that, and with a coefficient of 1, it remains so).

2) Algebraic polynomial functions can be decomposed into irreducible components, and evaluated as deterministic functions.

a) An irreducible component shall be defined as a sum of products of single-variable polynomials such that each variable is contained (if not but trivially) in exactly one factor of each term.

b) Furthermore, no term of an irreducible component shall have a preceding coefficient (should there be one it shall be absorbed by any one of the factors), and no factor shall be raised to a power (should such a factor exist, that factor shall be further reduced)

c) Take for example, F = 3(1+3R)4 + 2(1+P2+2P3)3(2+4Q+Q4)2 under modulo 5. This can be reduced to the following:

i) F = 3A4+B

ii) A=1+3R and B=ST

iii) S = 2M3 and T = N2

iv) M=1+P2+2P3 and N=2+4Q+Q4.

d) The following are the deterministic polynomials of the irreducible components:

i) F= [0+3A+3A2+3A3+3A4] [1+1B+1B2+1B3+1B4] +

[1+1A+1A2+1A3+1A4] [0+1B+2B2+3B3+4B4]

ii) A= [1+4R+2R2+0R3+3R4]

B= [0+1S+2S2+3S3+4S4] [0+1T+2T2+3T3+4T4]

iii) S= [0+2M+1M2+4M3+3M4]

T= [0+1N+4N2+4N3+1N4]

iv) M= [1+4P+1P2+4P3+0P4]

N= [2+2Q+1Q2+0Q3+4Q4]

e) These deterministic polynomials can be recombined when specific data is to be retrieved. The value of a deterministic polynomial in a single variable is the coefficient of the term whose degree is the value to be substituted for that variable. As values are determined, they can be algebraically manipulated via the structure of the deterministic function in which they reside (sums and products map to sums and products).

This being said, deterministic polynomials and functions allow all valuations to be represented simultaneously. Furthermore, they form an Alethic space A(VF, \*, *B*, *L*):

VF is the vector field described by prime modulus

\* is the operator that translates an algebraic polynomial function into its deterministic form

*B* defines the allowable algebraic manipulations of a deterministic function

*L* denotes the deterministic (logical) search algorithm (associates coefficients with powers).

All in all, the features of deterministic functions and polynomials exhibit the incredibly powerful nature of Alethic Data. Of course, more could be said on this subject, but I could spend an entire volume doing so. Therefore, I shall at this point bring Alethic Data to a close, and I shall leave you with avenues in which it could be extended.

# References

[1.1] Gallian, Joseph A. *Contemporary Abstract Algebra*. Boston, MA: Houghton Mifflin, 2006. Print.

[2.1] Hazewinkel, Michiel, Nadezhda Gubareni, and Vladimir V. Kirichenko. *Algebras, Rings and Modules*. Vol. 1. Dordrecht: Kluwer Academic, 2004. Print.

[2.2] Hazewinkel, Michiel, Nadezhda Gubareni, and Vladimir V. Kirichenko. *Algebras, Rings and Modules*. Vol. 2. Dordrecht: Kluwer Academic, 2004. Print.

[3.1] Tignol, Jean-Pierre. "Central Simple Algebras, Involutions and Quadratic Forms." *ChinaLectures.pdf* (1993): 13-17. *Lectures at the National Taiwan University, April 1993*. Web.

[4.1] Eisenbud, David. *Commutative Algebra: With a View toward Algebraic Geometry*. New York: Springer, 2004. Print.