

The General Theory and Method of Binary Logical Operations

Jeremy Horne
205 South Higley Road, No. 185
Mesa, Arizona 85206
E-mail: jhorne1@cris.com

Keywords: logic, operators, order

This paper sets forth the general theory of binary logical operations and the method for generating the logical space within which those operations occur. Included is a canonization of that generation and reasons for it. Underpinning the theory and method is a philosophy explaining why the logical space is generated in the manner described. The theory and method, combined with its philosophy, will provide a foundation for future developments in binary logic.

Logical Spaces

The Origin of Logical Space

This paper sets forth the general theory binary logical spaces and the method of performing binary logical operations within them. Our logical universe, or logical space, is generated serially and in ascending magnitude. Its origin is the existent and process. Logical space starts by accommodating a place for the existent, itself, the most basic object of the logical universe. Both process and its object as the existents, movement and stasis, constitute space-time.

Bound up in process is differentiation, a this and a that, from here to there, a displacement. Within process and out from it is generated the inchoate. Process then allows the inchoate to manifest itself. The inchoate is the evidence of process, and this evidence exists. Also, carried within the inchoate is process. However, the inchoate also must be manifest, and *its* manifestation exists. Because of differentiation, both generation, evidenced by the inchoate and its manifestation now have their others as referrers. The inchoate refers to its manifestation, and the manifestation refers to the inchoate. The existents, the inchoate and its manifestation, identify each other and themselves. Process and object then identify each other. In the physical world, displacement demarcates the object and gives it its character. It is not that a wave “collapses” in the quantum world, but it is the length of time that we observe a section of that wave that makes the object appear. Otherwise, the wave is movement.

This said, single letters, such as p, represent aspects of process, and digits (0 and 1 in our world) represent the existents. The inchoate is represented by 0 and its manifestation is represented by 1.

Table 1. The Existent

p
0

Both the existent and its other identify each other, and each is atomic. Everything else is a compound of either or both of these. (Note: Both, 0 and 1 have nothing necessarily to do with truth or falsity of propositions in this rendition of logic, although there are some parallels and crossovers.)

Table 2. The existent and its identifier

p	q
0	1

As we concern ourselves here only with the binary world, we also can say that this table completely expresses the existents, i.e., it is the Table of Existent Completeness.

Relationship

Differentiation is the foundation of relationship. This identifies itself through its other, that. That also identifies itself. Then, this is to that, and that is to this.

Now, with respect to an existent, there stands the:
 existent as itself, related to itself;
 existent with respect to its other, related to its other;
 existent's other respect to the existent, or other related to existent;
 existent's other as itself, the other related to itself.

As the p represents the generator, q as the identifier shall be its other, also a generator.

Table 3. The existent and its other, and the generation of the possible states of the existents together

p				
0	0	0	1	1
1	0	1	0	1

It says of the field of relations that the existent and its other may be related in maximally four distinct ways. Further horizontal arrangement of the existents 0 and 1 would be a repetition of what has already been stated. These are the permutations of relations.

This description has the formalism of letter pairs, such as pq, generating relations.

However, the two existents relate in four ways according to specific or definite order set by succession, the formal rendition of which is by the Table of Relational Completeness.

Table 4. Table Of Relational Completeness


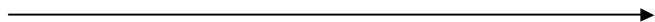
p	q	Relationship
0	0	existent as is related to or contained within itself.
0	1	existent as is related to or containing its other
1	0	existent's other as is related to or containing the existent
1	1	existent's other as is related to or containing itself.

Logical space canonization does not result from an arbitrary process; it is not an invention, but follows naturally as ordering expressed in the space-time (S-T) manifold:

Table 5. Space-Time Rendition of Relationships

Element P	Element Q	S-T axis
0	0	0s are parallel - contains itself
0	1	0 precedes, or contains
1	0	0 succeeds, or is contained
1	1	1s are parallel - contains itself

Table 6. World-Time Line of Existents

		Space-Time axis	World Line
0		parallel (Contained Within Itself)	
0			
0	1	Precedes (contains)	
1	0	Succeeds (contained)	
1		parallel (Contained Within Itself)	
1			
Time Line			

This is to say that an existent may come before or after the other or be parallel to another existent.

Space-time is what distinguishes existents, for 0 cannot be parallel to 1 or vice versa. Distinction is a product of the dialectic, and, perforce, parallel distinctions within the same existent are not possible. They are contained within themselves. Something cannot precede and succeed at the same time.

To this point, we know of the atomic existent. PQ has causes p to generate 0011 as a compound existent, and q to generate 0101 as a compound existent. The 0101 identifies in terms of 0011

and vice versa. This identification of compound existents in terms of each other establishes the foundation for generating other compound existents. That generation is accomplished by the means of the function. The complete display of these existents is the *Table of Functional Completeness*, discussed below.

The Development of a Function

The two compound existents are the bases for generating the other compound existents. (Note: As somewhat of a digression, the two compound existents may take other forms, such as 1100 and 1010, 1010, 1100, etc., but there must be a complete expression of possibilities, or permutations, of relationships. The present description follows from a serial or successive basis for generating existents, as reflected by the section above, *The Origin of Logical Space*. Is this development vital to the mechanics of calculating logical relationships? Surely, not, but the ordering principle preserves consistent systematic development and is conformant to how the logic maps to the real world. (Horne.) There are minimally two compound existents and they may be related only in a certain number of ways.

Functions designate ways of relating functions to each other, as well as the results of relationships. Binary-valued logic (binary logic, for short) relates two compound existents and the result is another compound existent. For compound existents (functions) the function operating over that relationship produces a third entity, also a function. The particular function number shows that the existents are related in a specific way for that function. The result is unique for that function.

The four rows of permutations of existent relationships yield a 16-column space, known, for our purposes, as the Table of Functional Completeness. This is generated by the same method as with the above tables - serially and in ascending order (binary counting), in this case vertically read from 0000 to 1111. Columns are headed by numerals 0 through 15 to designate the particular function. While Table 5 includes the p and q generators, they could be omitted, leaving the functions as operators, as well as objects. In this reference space, f_3 is always in the p column and f_5 in q column. (One possible semantic may be the space as bounded by degrees of knowing from totally unknown to totally known.)

Table 7. Table of Functional Completeness - total possibility of relationship between the two compound existents (something and its other)

p	q	f ₀	f ₁	f ₂	f ₃	f ₄	f ₅	f ₆	f ₇	f ₈	f ₉	f ₁	f ₁	f ₁	f ₁	f ₁
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0

NOTE: f^* is an object, as well as a process (operator). The combination of 0s and 1s imparts the ascendancy quality, while 0s and 1s by themselves denote existence.

General Method of Establishing Functional Relationships

How to read and use the table

Functions in this system have four parts, each part operating over a row of the possible space described above. For example, referring to atomic existents, row 3 for function 4, is read, “function 4 relates 1 to 0 (which is always row three for possible space) to yield 0.” Compound existents as functions are related by reading all four digits of the function vertically and operating over two other functions to yield a third. Each row of the function is a "deductive instance," where there is a description of a specific relationship between two points in space-time. The third row, for example, has the deductive instances of 1 and 0 to yield 1.

As an example, in treating function 4 as a compound existent, read “function 4 operates over function 3 and function 5 to yield function 4.” That is, any function operating over functions 3 and 5 will yield itself; i.e., it defines itself. This table contains any four binary digit function as an operator, argument, or outcome, thus making the system closed and deductive.

As another example, allow f_9 to operate over p and q . The resulting generation is the space designated by 1001. In other words, 1001 characterizes f_9 , as the following standard canonization illustrates.

Table 8. **Generation of functions – example of f_9**

p (f_3)	<-> (f_9)	q (f_5)	Generated resultant f_9 in the field
0	1	1	1
0	0	1	0
1	0	0	0
1	1	0	1

Thus, the general tabular schematization for functional generation is:

Table 9. **Truth Table of A Function Operating over f_3 and f_5**

p (domain) (f_3)	f^*	q (range) (f_5)	Generated resultant in the field – the function, itself, f^*
0		0	
0		1	
1		0	
1		1	

General Functional Relationships

Now that we have seen how the operators are generated, how are existents related starting with *any* initial possible space?

For example, in standard truth table canonization, we have as an example of function instances f_2 (0010) and f_{12} (1100) related by f_9 (1001), commonly called "equivalence," the resultant being f_1 :

Table 10. Example of f_9 operating over f_2 and f_{12} to generate f_1

p (f_2)	<-> (f_9)	q (f_{12})	Generated resultant f_1 in the field
0	1	1	0
0	0	1	0
1	0	0	0
0	1	0	1

Modifying Table 9, we have as a schema:

Table 11. Tabular schema for generating resultant in field

p (domain)	f^*	q (range)	Generated resultant in the field
v^*		v^*	
v^*		v^*	
v^*		v^*	
v^*		v^*	

where v^* is either 0 or 1.

About the Table of Functional Completeness Generally

The functions, or operators, both process information in logical space, and are, themselves, units of information. A function is a composite of a proof in that p and q, or the variables, are placeholders for functions. as in $p \vee (q \supseteq r)$, where the number of rows in the tabular rendition of the expression equals 2^n , n being the number of variables. With $p * q$, the number of rows equals four, where * is any function consisting of the four bits representing particular deductive instances. As soon as one determines the number of placeholders for values, the size of logical space is automatically determined, as well. However, the 4 rows by 16 columns logical space is the basic building block of these larger logical spaces, as may be seen by inspection. There are two types of logical space: pre-determined and sequentially ordered by the number of variables (as illustrated above), and space resulting from operations (such as a "truth table" computation). In a formal deductive proof, there are premises, as coupled functions by the means of inference and equivalence rules may be derived a conclusion, another expression.

Canonization

Standard truth table canonization computation is adequate for displaying relationships. It is even helpful. However, the above logic is not truth table logic, but an epistemological one; it displays how we know. (Horne) What follows, such as functions of functions, the canonization immediately becomes unwieldy, necessitating shorthand and more manageable functional notation more closely in line with the conceptual development of this method. (e.g.: See Copi, Exner, Klenk, and Massey, *passim*.)

Where i and j are two functions, only the functions are expressed as operands, rather than the compound existents. Note that the operands can be operators.

In our canonization:

f^* is a function operating over two vertical strings of 0s and 1s as compound existents making up functions. This arrangement yields a third function, i.e.,

$$f^*(f_i, f_j) = f_n$$

As an example:

$$f_9(f_2, f_{12}) = f_1$$

means:

$p (f_2)$	$\leftrightarrow (f_9)$	$q (f_{12})$	Generated resultant f_1 in the field
0	1	1	0
0	0	1	0
1	0	0	0
0	1	0	1

Functions as homeostats

Each of the 16 functions is homeostatic, as reflected by recursively feeding functions outputs back in itself. If the operator is continuously "fed" two functions at a time without regard to previous output, there will emerge a logical space with a pattern, as illustrated by Wuensche and Kauffman with randomly coupled functions and Basins of Attraction (Kauffman, Chapter 5, and Wuenche). Since each output is binary, it will be found as a column in logical space, hence contained within the system and deduced. It is not new information.

Now, we can place restrictions on how the information is given to the operator. By feeding the outputs back to the four-digit operator as inputs, the outputs eventually will be repetitive, and the processing by the function will stabilize. Similarly, a deductive logic proof seeks stability. The proof will accept information (other functions) as premises and process them, ultimately reaching a goal state of stabilization, where the output is a repetition of the input. Feeding this

input back into the proof as premises simply repeats the proof. In both cases, there is no longer produced any new information, there being stabilization, the function(s) having reached the goal or deductive state. A full complement of information has been processed by the function. A system (be it a single operator interacting with other operators or a proof) that interacts with its environment and maintains itself in a state of equilibrium is called a homeostatic automaton. The next section describes its mechanics.

Structure and process of the binary homeostatic automaton

A function operating on two other functions outputs another function found in generated, discrete, and subsequently bound two variable logical space. To display the homeostatic properties of the function, recursively feed back in the following manner the outputs as inputs until the function starts repeating outputs.

1. $f_2(f_3, f_5) = 0010 = f_2$

2a. $f_2(f_2, f_5) = 0010 = f_2$. The "p" half terminates, since the output of f_2 is a repetition of a previous output, f_2 , and its reprocessing as an input obviously will result in another repetition of f_2 .

2b. $f_2(f_3, f_2) = 0001 = f_1$ (Note that the order to be evaluated is f_3, f_2 , and NOT f_2, f_3)

3a. $f_2(f_1, f_5) = 0000 = f_0$ This half now is f_0 , by virtue of 2b and continues on with $f_2(f_0, f_5)$ and $f_2(f_5, f_0)$.

3b. $f_2(f_3, f_1) = 0010 = f_2$ This half terminates with f_1 , by virtue of 2b.


The function is unstable ultimately for only four iterations.

A state diagram exhibiting its homeostatic behavior may represent each operator. For example, these are the state diagrams for the f_7 and f_9 homeostatic functions. The | indicates termination of a recursion, or iteration, i.e., when the outputted function is a repetition of a previously outputted function.

f_7

+ ↑ Branchings ↓ -		Iteration 1
		$f_7(f_7, f_5) \rightarrow f_7$
	$f_7(f_3, f_5) \rightarrow f_7$	
		$f_7(f_3, f_7) \rightarrow f_7$

f₉

		Iteration 1	Iteration 2	Iteration 3
		f₉(f₃,f₅) → f₉		f₉(f₅,f₅) → f₁₅
	f₉(f₉,f₅) → f₃		f₉(f₁₅,f₅) → f₅	
				f₉(f₃,f₅) ⇒ f₉
		f₉(f₃,f₃) → f₁₅		
				f₉(f₃,f₅) → f₉
			f₉(f₃,f₁₅) → f₃	
				f₉(f₃,f₃) → f₁₅
		f₉(f₅,f₅) → f₁₅		
	f₉(f₃,f₉) → f₅			
		f₉(f₃,f₅) → f₉		

How many of iterations and much information or logical space the operator consumes ("sub-functions" generated by each iteration) will vary with the operator. So, f₁₃ may process f₃ and f₅ recursively for 3 iterations and terminate, but f₈ might take 7 iterations. The same initial information is processed, but the state diagram shows the other areas of logical space involved, thus giving the function an "anatomical description" of its attempt to gain homeostasis. The two vector components are the number of divergences at each iteration (node creations) and number of iterations required to reach equilibrium, the point where the function is stabilized and starts repeating its outputs. Each operator has a different complexity and can be ordered according to its vector descriptions. In terms of computational efficiency, a function has to process a quantity of information (or it only has to process that much less information) to maintain itself as a homeostatic automaton.

To symbolically describe a function, one might take the Cartesian product of the differentials of both the X and Y components of the function, i.e., take the differential of each of the X and Y components to describe specifically the behavior of iterations and branching, respectively.

$$\text{Branching} = Y = \int I dx/dy \text{ (for 0 to n upward branching)} + \int II dx/dy \text{ (for 0 to n downward branching) and}$$

$$\text{Iterations} = X = \int B dy/dx, \quad \text{where } I = \text{upward, } II = \text{downward, and } B = \text{branching}$$

To completely describe that function's behavior, we have the Cartesian product X × Y.

A question is whether the same efficiency exists for the function acting recursively over any two initial functions other than f₃ and f₅.

Larger Discrete Binary Spaces

If micro logical spaces re-emerge as themselves after repeated auto feedback, do larger discrete binary spaces also have the same characteristic? A caveat exists here – this refers to discrete spaces bounded by a regular column, or Y-axis, i.e., multiples of four existents. Above, p generated the compound existent 01. That is, each four-digit unit is a function. Partial functions acting in this recursive manner yield partial functions that exhibit a range of possible feedback loops. The resulting spaces still are closed, albeit fringed by bounded possibilities. However, this discussion lies outside the scope of this paper and will be discussed in future works.

A discrete binary space is an n-dimensional bounded area in which each component assumes exclusively either of two values, conditions, or states. Thus, each of the 16 functions discussed above occupies a discrete binary space. In larger discrete spaces, randomly generated bit streams display "basins of attraction," according to Wuensche and Kauffman, as cited above. Because such basins are repetitive and are comprised of binary functions, it would be reasonable to ask if the recursive characteristics of those functions discussed above would also apply to these spaces. The reason feedback four bit functions repeat themselves may be the same as for the more than four bit functions and in larger spaces. As an aside, the epistemological semantics given to the values, as well as the nature of the deductive process adds a philosophy to the search.

A discrete binary space as a homeostatic automaton processes its environment, thereby producing outputs. The outputs, ipso facto, change the character of the environment. In turn, the discrete space processes the changed environment, and, in the course of doing so, tries to maintain itself. After a number of iterations, if the initial space is homeostatic, it will stabilize, evidenced by returning to its initial condition. That is, recursion stops upon re-iteration of the space. Note that these spaces as structures may contain anything mappable to them, such as computer programs, sunspot activity, or automatons.

Essentially, the methodology is the same for examining any n-dimensional space recursively as is with the individual functions. Following is an informal presentation of this approach to spaces of two or more dimensions. It is not meant not to be a complete discussion but illustrative.

The environment for an n-dimensional space consists of the permutations of bits of that space. For example each function as a one dimension has as its environment, 2^4 or 16 permutations. An i by j matrix has 2^{ij} permutations as its environment.

To examine the recursive character of a space:

1. $f^*(AS, PS) = NS$, where f^* is one of sixteen functions, AS is the initial space being examined, PS is one of the possible spaces, and NS is the resulting space. In matrix notation, where AS, PS, and NS are matrices: $|AS| * |PS| = |NS|$.

2. Recursively:

2A. $f^*(NS, PS) = NS$, or $|NS| * |PS| = |NS|$ first branch

2B $f^*(PS, NS) = NS$, or $|PS| * |NS| = |NS|$ second branch

for each of the 16 functions, resulting in $16(2^j)$ generations.

3. The question at this point is whether the attractor space is recursive in this manner and based upon the same principles as the individual function space. If so how can the information processing efficiency of each operator be used to disassemble the recursive process within the larger spaces and the boundaries between patterned and chaotic space?

Application

If a random number generator (RNG) produces an attractor space, we can “dissect” that space by looking at the complexity of the recursion space after each iteration and comparing it to operator prioritization based upon information processing efficiency, we may be able to explain the steps of each iteration. (Horne.)

Of course, if the RNG does generate patterns, it is suspicious that it is random, and the recursion spaces may reveal this.

For example:

$$Fn(fITS, fAS) = f(OS)$$

$$Fn(fOS, fAS) = fOS \quad \text{and} \quad fn(fITS, fOS) = fOS$$

Where,

Fn is one of the 16 functions. Note: any n-ary space is reducible to 1-D space. eg: pqr, is expressed in terms of 1D space - i.e., the 16 operators.

ITS = One of total possible spaces or permutations of spaces

AS = Attractor space - particular state space in which attractor is found

OS = Output space - resulting from a one of 16 functions acting over TS and OS.

In addition, apply a recursion methodology, such as above. Either calculate individual columns of the whole space with each of the operators, or the whole space, itself.

I have presented an approach to showing how discrete binary spaces may reproduce themselves and how we may “dissect” those spaces based on recursion. Others have shown that randomized space contains basins of attraction with “handedness,” or chirality (Kauffman, and Wuensche). Chirality appears manifest in the recursion of four-bit functions, as well. Since any two-

dimensional (2D) space is a composite of 2 variable or 4 bit functions, it would not be unreasonable to suggest that attractors have their origin in the recursive and chiral nature of four-bit functions.

If operators have a natural order and may be grouped according to information processing efficiency, as suggested by their recursion graphs, then what of the character of binary spaces comprised of such functions? It would appear that some spaces are simpler than others are in terms of informational complexity. (Horne.)

Heuristics, parsing methods, and other devices would determine various pattern recognition schemes. Patterns emerging from my work, however, would be generated from logical operations based upon an empirically derived ordering of operators. The reason fed-back four bit functions repeat themselves may be the same as for the more than four bit functions used by Wuensche et al. The reasons lie in the nature of deduction

We know that neural nets operate in groups to process information, and that a temporal factor is operant in information processing. One might observe the patterns and compare to those generated by cellular automata, or electroencephalograms, as suggested by Wuensche. (Wuensche 1993, p.11. EEGs may be correlated with temporal coding in neural populations. "Temporal Coding in Neural Population?" 1997 275 *Science* p. 1901). If temporality is important in neuronal information and logical operations can be correlated to EEGs, then, logical operations are temporally bound. Elsewhere, I have argued that the values of 0 and 1 are representations of state collapse of wave functions. That is, "truth tables" are descriptions of wave function collapse, there is a neuronal correlate, and there are specific reasons for this. (Horne)

As can be seen above, there is a philosophical and theoretical side to the structure of binary logic. It is a bit more than a crude applications device for ordinary language translation.

References (Volume Journal Page, Date):

_____, "Temporal Coding in Neural Population?" 1997 275 *Science* p. 1901.

Copi I. (1979) *Symbolic Logic*. New York: Macmillan Publishing Co.

Exner R.E., Myron F. Roszkopf (1959) *Logic in Elementary Mathematics*. New York: McGraw-Hill Book Company.

Hameroff S. and Roger Penrose (1996) "Orchestrated Reduction of Quantum Coherence in Brain Microtubules: A Model for Consciousness." *Toward a Science of Consciousness: The First Tucson Discussions and Debates*. Eds. S.R. Hameroff et al. Cambridge, MA: MIT Press. p. 507-540.

Kauffman S. *The Origins of Order* (1993) New York: Oxford University Press.

Hameroff S. and Roger Penrose (1996) "Orchestrated Reduction of Quantum Coherence in Brain Microtubules: A Model for Consciousness." *Toward a Science of Consciousness: The First Tucson Discussions and Debates*. Eds. S.R. Hameroff et al. Cambridge, MA: MIT Press. p. 507-540.

Horne, J. "Logic as the Language of Innate Order in Consciousness." 22 *Informatica*,. 4, December 1997.

Klenk, V. (1989) *Understanding Symbolic Logic*. Prentice-Hall. Englewood Cliffs, NJ.

Margaris, A. *First Order Mathematical Logic*. Waltham, Massachusetts: Blaisdell Publishing Company.

Massey, G.J. (1970) *Understanding Symbolic Logic*. New York: Harper & Row, Publishers.

Wuensche, A. (1993) *The Ghost in the Machine: Basins of Attraction of Random Boolean Networks*. (University of Sussex at Brighton: Cognitive Science Research Papers, CSRP 281.