

# Recursion of logical operators and regeneration of discrete binary space

Jeremy Horne  
 15 Copper Hill Ct.  
 Durham, NC 27713  
 E-mail: jhorne1@cris.com

**Keywords:** logic, recursion, operators, order

**Edited by:** Anton P. Zeleznikar

**Received:** September 11, 1999

**Revised:** February 18, 2000

**Accepted:** March 15, 2000

*Any discrete (closed) binary, or Boolean, space is recursive. That is, if the outputs of functions are repeatedly forward-fed into those functions, those outputs will present themselves again for processing. That is, the full functionality of an operator reproduces itself. Each of the 16 operators, or functions, in a two variable system is a self-maintaining (homeostatic) automaton in logical space. The homeostatic character of the function is displayed by that recursion. As larger binary spaces are comprised of the functions (partial or complete), functional recursion may open the way to analysing basins of attraction in spaces produced by the random concatenation of operators to reveal the character of pattern generation. Further, recursion of binary logical operators may have correlates in biological neural networks.*

## 1 Synopsis

Any discrete (closed) binary, or Boolean, space is recursive. That is, if the outputs of functions are repeatedly forward-fed into those functions, those outputs will present themselves again for processing. That is, the full functionality of an operator reproduces itself. Each of the 16 operators, or functions, in a two variable system is a self-maintaining (homeostatic) automaton in logical space. The homeostatic character of the function is displayed by that recursion. As larger binary spaces are comprised of the functions (partial or complete), I suggest that functional recursion may open the way to analysing basins of attraction in spaces produced by the random concatenation of operators to reveal the character of pattern generation. Further, recursion of binary logical operators may have correlates in biological neural networks. Two issues emerge about how to approach pattern analysis in binary space.

First, the way each operator renders an environment displays its information processing character and efficiency. The character of a function's recursion is revealed by its outputs, and efficiency is measured by how rapidly an operator manages the complexity of the other functions it processes. (There is no issue of accuracy in such a measurement, since the outcome is deterministic.) Both character and efficiency may be components in discovering the sources of patterns in basins of attraction or deciphering what appears to be noise in digital space.

Second, prioritisation schemes in a parenthesis-free notation might be based on such an efficiency in order to

measure overall computational efficiency in the system. Ranking of operators in a parenthesis-free (ungrouped) expression based on information processing efficiency also may be a means for exploring the basis of efficient logical thinking, or how fast a person can process logical alternatives to arrive at a correct conclusion.

## 2 The system's syntax – Structure of the function

There are 16 operators generated from a binary relationship schema, each operator being a function with a specific degree of complexity, as will be discussed informally below. We obtain descriptions of functions with the permutations of symbols, such as 0 and 1 (00, 01, 10, and 11), resulting in a logical space (table of functional completeness) in the form of a 4 row by 16-column table, with each column headed by functions  $f_0$  through  $f_{15}$ . I use the symbol set {0,1} to indicate columns that start with 0000 under the  $f_0$  operator and end with 1111 under  $f_{15}$ . This would be 0 through F in hexadecimal.

$f_0$	$f_1$	$f_2$	$f_3$	$f_4$	$f_5$	$f_6$	$f_7$	$f_8$	$f_9$	$f_{10}$	$f_{11}$	$f_{12}$	$f_{13}$	$f_{14}$	$f_{15}$
0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Figure 1: Table of Functional Completeness.

All 16 relationships display how the first element is related to the second for each permutation of 0 and 1. For example, 0011 (the "0" often regarded as a "false," and 1 as "true") can be related to 0101 as 0111, and we call the relationship "or" ( $0 \text{ or } 0 = 0$ ,  $0 \text{ or } 1 = 1$ , etc.). Each function, then, is an ordered set of four elements. Note also that 0011 (or  $\{0011\}$ ) is  $f_3$  and 0101 is  $f_5$ . In standard truth table form using placeholders p and q for the functions  $f_3$  and  $f_5$ , respectively and systematically displaying all four permutations of 0 and 1 in a binary relationship,  $f_7$  is:

p	q	p or q
0	0	0
0	1	1
1	0	1
1	1	1

Figure 2: Permutations of 0 and 1.

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.

The functions, or operators, both process information in logical space, and are, themselves, units of information. Alpha characters are placeholders, or the variables, for functions, as in  $p \vee (q \supseteq r)$ , where the number of rows in the "truth table" rendition of the expression equals  $2^n$ , n being the number of variables. With  $p * q$ , the number of rows required to describe a function completely equals four, where \* is any function consisting of the four bits representing particular deductive instances.  $P * q * r$  requires eight rows;  $p * q * r * s$  needs 16 rows, and so forth. 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: predetermined 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, premises, including expressions of coupled functions, imply conclusions by the means of rules (axioms, inference rules, and equivalences). It must be borne in mind that every premise, intermediate expression, and conclusion in a proof is a function.

### 3 The system

The system in which the functions, or operators, perform is closed, and both functions and system seek to maintain integrity. Disorder, the breakdown in explicit placement of elements, results when the system accepts information not already found within its universe; certainty turns to probability. Deduction is a closed process, where any function or coupling of functions as a rule or rules, and entities upon which those rules act, yields a result

predetermined by that coupling. (Recall that a function also is information.) In common parlance, "if the premises are true, the conclusion must be, as well." There are two aspects of deduction. First, there is system deduction, where an introduction of information outside the system, such as a function other than the ones within the system or an arbitrary extension of logical space with other functions disorders the system. A second aspect concerns maintaining the integrity of structures within the system. This could be called subsystem deduction.

### 4 Functional Recursion

If the operator is continuously "fed" two functions at a time without regard to previous output, there will result a logical space with a pattern (Wuensche and Kauffman, Chapter 5). Since each output is binary, it will be found as a column in logical space, hence contained within the system and deducible. It is not new information.

Emerging patterns from these ostensibly randomly coupled operations in a closed system suggest a way of discovering the nature of that reputed randomness. If we can place restrictions on how the information is given to the operator, we can see that the repetitive character of the operators within the space shows functional self-maintenance.

By feeding the outputs back to the four-digit operator as inputs, the outputs eventually will be repetitive, meaning that the function has stabilized. A systems analysis view provides a reason for this. Think of a binary relationship, such as p & q as being an atomic proof, that is, " $\{0011\}$  and  $\{0101\}$ . Therefore  $\{0001\}$ ." A deductive logic proof seeks stability. In the case of self-maintenance, 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. For both the function and proof, 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.

### 5 Structure and process of the binary homeostatic automaton

Keeping with our designation of p and q variables as placeholders for functions  $[f_*(p, q)]$ , the recursion, using  $f_2$  as an example works in the following manner. (Note that each function is an ordered set.)

$$1. f_2(f_3, f_5) = f_2: \{0010\}(\{0011\}, \{0101\}) = \{0010\}$$

2a.  $f_2(f_2, f_5) = f_2$ .  $\{0010\}(\{0010\}, \{0101\}) = \{0010\} = \{0010\}$  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.  $f_2(f_3, f_2) = f_1$ .  $\{0010\}(\{0011\}, \{0010\}) = \{0001\}$  (Note that the order to be evaluated is  $f_3, f_2$ , and NOT  $f_2, f_3$ )

3a.  $f_2(f_1, f_5) = f_0$ .  $\{0010\}(\{0001\}, \{0101\}) = \{0010\} = \{0010\}$  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.  $f_2(f_3, f_1) = f_2$ .  $\{0010\}(\{0011\}, \{0001\}) = \{0010\}$  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 behaviour may represent each operator. For example, these are the state diagrams and graphs for the  $f_7$  and  $f_9$  homeostatic functions.

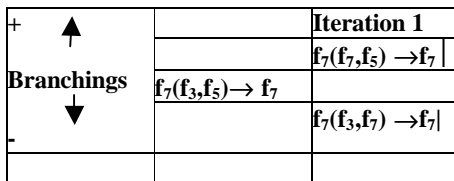


Figure 3: State Diagram for  $f_7$ .

Every time a set of p and q variables is used, there is created a branch, or divergence. In  $f_7$  there are two branches, one for the  $f_7$  function in the p placeholder, and the other for the  $f_7$  in the q placeholder. The graph for  $f_7$  is:

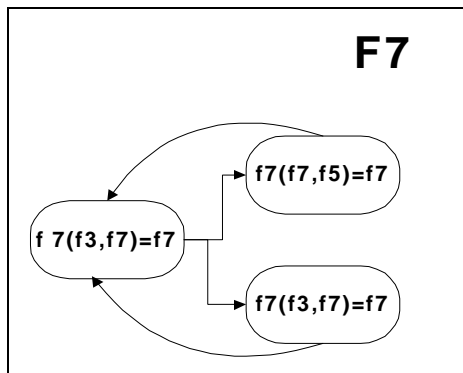


Figure 4: Graph for  $f_7$ .

The state diagram for  $f_9$  is:

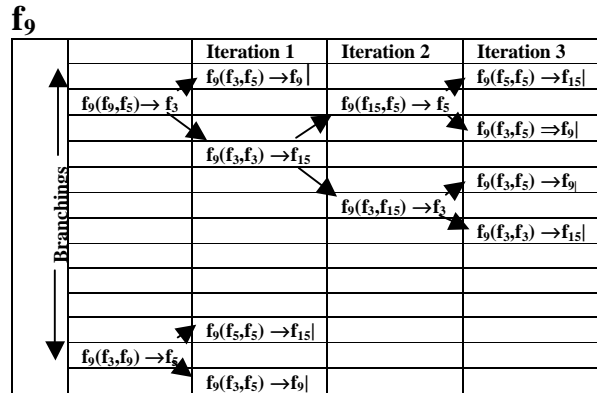


Figure 5: State Diagram for  $f_9$ .

In  $f_9$ , there is one branch for the p side of the diagram, but there are ultimately three branches, or divergences on the q side. The graph for  $f_9$  is:

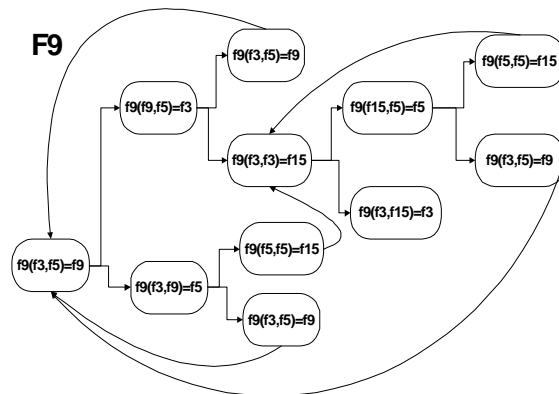


Figure 6: Graph for  $f_9$ .

Each of the 16 functions is recursive, i.e.,  $f_*(f_3, f_5)$ . Furthermore any function over any other two functions,  $f_*(f_n, f_p)$  is recursive, as well, because there can be only a maximum of 16 iterations (only 16 unique functions) on any branch before there is a repetition of one of the previous. Perforce, even if there are 16 unique outputs on a branch, there is repetition on the 17<sup>th</sup> iteration. Because of this, no function can have more than sixteen iterations.

Diagrammatically,

For the p side: $F^*(f_p, f_q) = f_a$ $F^*(f_a, f_q) = f_b$ $F^*(f_b, f_q) = f_c$ .... Max $F^*(f_p \leq 16, f_q)$ before there is repetition	For the q side: $F^*(f_p, f_q) = f_a$ $F^*(f_p, f_a) = f_b$ $F^*(f_p, f_b) = f_c$ .... Max $F^*(f_p, f_q \leq 16)$ before there is repetition
---	---

Figure 7: Maximum Iterations is 16.

The number of times a function receives input before it repeating the output (number of iterations) and how much information or logical space the operator consumes (number of "sub-functions" generated by each iteration) will vary with the function. So,  $f_{13}$  may process  $f_3$  and  $f_5$  recursively for three iterations and terminate, but  $f_8$  might take seven 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 branches, or 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.) A question is whether the same efficiency exists for the function acting recursively over any two initial functions other than  $f_3$  and  $f_5$ .

## 6 Recursion of Discrete Binary Spaces

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 occupy a discrete binary space. In larger discrete spaces, randomly generated bit streams display "basins of attraction," or repeating functions, according to Wuensche and Kauffman. Inasmuch as such basins are repetitive and are comprised of binary functions, it would reasonable to ask if the recursive characteristics of those functions discussed above would also apply to these spaces and be useful in understanding the repetitions within those spaces.

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. There are three ways in which a discrete binary space can be shown to be recursive.

- First, since the space is composed of one or more of the 16 functions and because functions are recursive, we see that the space, then, is recursive.
- Second, because  $f_*(f_p, f_q)$  is recursive, any column of the space can act over the next two columns recursively.

- Third, 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:  $|AS| * |PS| = |NS|$ .

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  $16(2^{ij})$  generations.

## 7 Character of Attractors – Starting point for analysis

Whenever a function produces an output previously produced, there is begun a cycle that has been previously iterated. This repetition point forms the basis of an attractor. Every function operating over two other functions results in one of more emerging attractors. While it seems that logical operators are randomly coupled to produce patterns, it would be interesting to see what patterns emerge if the operators were coupled according to an empirically determined scheme based upon a specific ordering of operators. (Horne, 1997)

A concept of information processing efficiency of each operator may be used to disassemble the recursive process within the larger spaces and the boundaries between patterned and chaotic space. To do this requires affixing an order of operators based upon a standard, a subject for further research. A starting point would be to examine how many iterations it would take for each function to process inputs  $f_3$  and  $f_5$ . From the above, the functions could be ranked according to the number of iterations required to process  $f_3$  and  $f_5$ . Thus,  $f_0, f_1, f_3, f_5, f_7, ,$  and  $f_{15}$  might be ranked co-equally and most efficient in maintaining themselves,, since each has only one iteration. Those functions having two iterations,  $f_{10}$  and  $f_{12}$ , would be the next most efficient, and so forth. Taking into account the number and type of branches could be a factor in a function's complexity and bear on how it processes its environment.

## 8 Ramifications and applications of theory – avenues for exploration

A discrete binary space is comprised of elements having one of two possible conditions, a factor of great import. Any phenomenon that can be digitised is subject to

recursive analysis. Both ordered and unordered phenomena can be digitised and set in a discrete binary space. Sunspot activity, waves, and background "noise" are candidates. Every piece of computer code, whatever the language, can be reduced to machine language, 0s and 1s, meaning that every computer program is a discrete binary space. To say that a discrete binary space is recursive is to say, also, that any computer program is recursive and can reproduce itself.

How would one determine whether there is order, or pattern, within any discrete binary space? There are attraction points found within autonomous random Boolean network (BN) state-space (Kauffman 1993, Chapter 5). With respect to ordering in Boolean complexity, it can be demonstrated that numerous random couplings of operations result in patterns resembling those of cellular automata (Wuensche 1993, *passim*). By "dissecting" these spaces by functional recursion, it may be possible to discover the nature of any attractors and dispel any notion that the patterns were generated "randomly." Too, spaces without any discernible pattern might also be analysed with recursion.

## 9 Summary

Any discrete binary space can repeat itself because it is composed of one or more of the 16 binary logical functions that are recursive. The reason fed-back four bit functions repeat themselves is the same as for the more than four bit functions, as the latter are composed of the former. (Partial functions – three or fewer digits - are repeatable, albeit the graphs are more complicated. Each possible outcome has to be graphed. {001}, for example would require graphs for {0010} and {0011}.) A discrete binary space as a homeostatic automaton processes its initial environment, thereby producing outputs that, *ipso facto*, change the character of the environment. 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 function (or space) is homeostatic, it will stabilize, evidenced by repeating outputs characteristic of how it processes the environment. Discrete binary spaces may be analysable with the recursion method; a larger space reputedly randomly generated and containing a pattern may be analysable with functional recursion and may not be random at all

A number of actual situations arise where binary grouping is at issue in analysing spaces. Some operators may be more efficient than others in the way they process information and maintain themselves in the environment. Logic and automata seem to describe how neural pulses behave (Hameroff and Penrose, *passim*) and a question is whether operational ordering would make a difference in how binary patterns emerge from the surface of neuronal microtubules. Basins of attraction may bear a relation to cellular automata, or electroencephalograms (EEG), as suggested by

Wuensche. (Wuensche 1993, p.11) EEGs may be correlated with temporal coding in neural populations, since neural nets operate in groups to process information, and a temporality enters in information processing. (Fetz p. 1901) If temporality is important in neuronal information and logical operations can be correlated to EEGs, then, logical operations are temporally bound. Further, neural networks "...exhibit emergent properties such as ... generation of distinct outputs depending on input strength and duration, and self-sustaining feedback loops." (Bhalla and Iyengar, p. 381) If mapped to biological neural structures, binary recursion as discussed above, becomes immediately relevant. To add to the space-time theme, I have argued elsewhere that the values of 0 and 1 are representations of wave function collapse, similar to "truth tables." (Horne.1997)

As can be seen above, there is a philosophical and theoretical side to the structure of binary logic. It is a bit (not only a pun) more than a crude device used for ordinary language translation.

## 10 References

- [1] Bhalla, Upinder S. and Iyengar, Ravi (1999) "Emergent Properties of Networks of Biological Signaling Pathways" *Science*, 283, p. 381-387.
- [2] Fetz, Ebherd E. (1997) "Temporal Coding in Neural Populations?" *Science*, 278, p. 1901-1902.
- [3] 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. MIT Press. Cambridge, MA, p. 507-540.
- [4] Kauffman S. (1993) *The Origins of Order* Oxford University Press. New York.
- [5] Horne J. (1997) "Logic as the Language of Innate Order in Consciousness" *Informatica*, Vol. 22, No.4, pp. 675-682.
- [6] Hayes P. J., Novak G. S. & Lehnert W. G. (1992) ACM Forum - In Defence of Artificial Intelligence. *Communications of the ACM*, 35, 12, p. 13-14.
- [7] Wuensche A. (1993) "The Ghost in the Machine. Basins of Attraction of Random Boolean Networks" in *Cognitive Science Research Papers, CSRP 281* University of Sussex at Brighton.