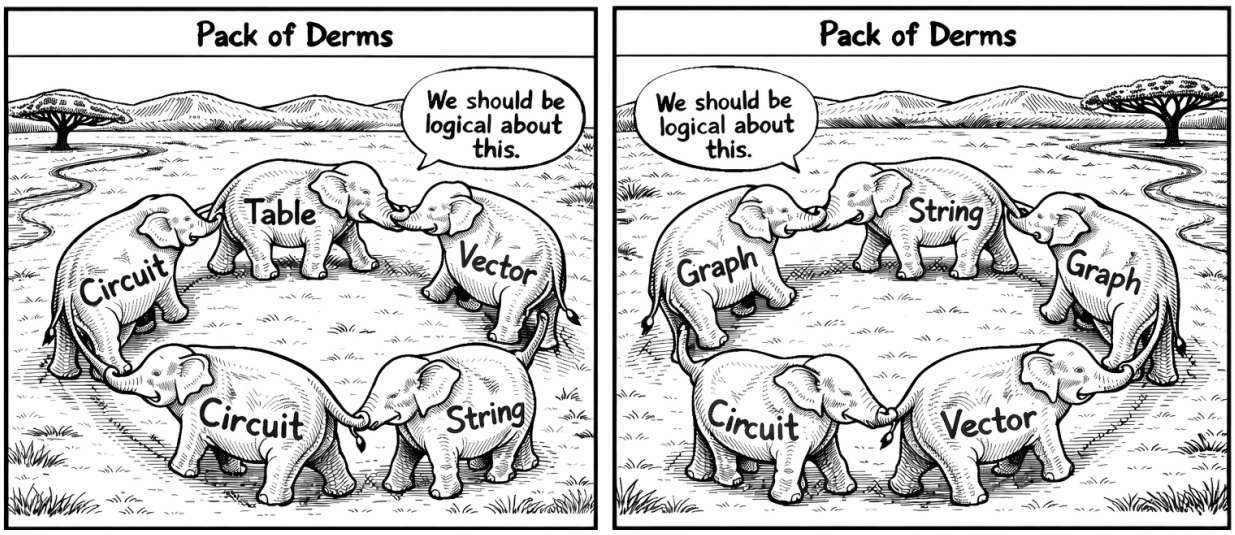
Chapter 26
"Engineering"



ChatGPT via Author 'Guidance': "Pack of Derms."

George Bool

Logicians and digital engineers base their fields, their entire methodologies, on Boolean Algebra, but in incompatible ways.

Logicians are interested in logical forms, expressions which must be either true or false, and in transformation steps (lemmas) which guarantee stepping securely from one true statement to another (proofs). They are interested only in statements which are true independent of inputs. They give them a formal name, *tautologies*. Self-reference is poison to them, and they go to great lengths to ensure it cannot intrude into their carefully crafted worlds.

Digital Engineers are also interested in logical forms, but in those which can be combined in interesting ways to do work. They put the inputs to use. To an engineer, a tautology is a wasteful circuit – an easy optimization. It is replaced with a tap to Vcc and erased from the silicon, freeing space for something useful.

The contrast is stark, logicians avoid self-reference while engineers embrace it. The *paradox* of self-reference is how they achieve *oscillation*, the steady tick-tock of the global clock that enables computation. They push that clock rate as high as the physics of silicon will permit. On the flip side it's the *indeterminacy* of self-reference that allows them to implement *memory*.

Two professional fields – highly impactful, grounded in the same fundamentals – but with divergent emphasis. Who'd a thought?

Poor George – he is spinning in his grave.

The engineers have mastered self-reference, the Logicians have not. Engineers call it feedback. To them it is like water to a fish. Logicians shun it because it makes a hash of equality, transitivity, and completeness. Their paradigm is the *proof*, the faith that by starting from true axioms and valid

rules, they can neatly step from truth to truth secure in the knowledge that their *formal system* prevents any mistake. They believed all truths could be revealed this way, that logic, mathematics, and science could be placed on unassailable solid ground.

Then Gödel showed they could not keep self-reference out. Worse, he showed that their formal systems must be either contradictory or incomplete. From within that paradigm, this was a nightmare.

Let's see why the engineer's world did not unravel. How self-reference was mastered not by exclusion but by application.

Road Map

As powerful as the *mark* notation is, and as brilliant as its ability to derive logic from first principles is (unique among formal systems as far as I know), it does suffer from a few limitations. The first is cultural; it is not a well-known formalism. Second, it is not a typographic system; while ideal for paper and pencil, it doesn't translate well to modern editing tools. Third, and most grievous, it cannot represent all possible self-referential forms and thus cannot be used to determine their total number or create a complete taxonomical characterization of them.

This chapter addresses those limitation by introducing five different tools of representing and analyzing self-referential logical forms. These five tools work well together and as a set both reinforce each other and address the limitations of the mark notation.

Before we begin it is worth reviewing why we are going to so much trouble. The impetus is the conflict between relativity and quantum nonlocality. Nonlocality implies FTL, which relativity says is impossible because FTL implies time travel. Given spacelike causality one gets both. The strongest argument against time travel is the grandfather paradox, the possibility of a causal paradox. The reason for pursuing imaginary truthvalues is that they resolve the problem of logical paradoxes by introducing conjugate bases for logic. QTP is simply the hypothesis that conjugate bases might resolve causal paradoxes in physics and do so in a way that solves the measurement problem. Put another way, the Universe allows time travel (at the quantum level, not at the classical level), but it comes with a cosmic censor that prohibits temporal paradox. The cosmic censor *is* the measurement process; an observer independent objective process that collapses an isolated quantum system in the one-and-only one basis in which it is not paradoxical.

Learning five things is not that hard, just tedious, they have to be learned one at-a-time. But in our case, they are isomorphic representations of the same thing. Knowing even one other, helps to learn the next one. To make the pedagogy as effective as possible, here is a road map of the five tools, along with brief notes on what they bring to the party.

The Five Tools

1. Digital Circuit Schematics – how computer engineers represent logic and self-reference (effortlessly), a **graphical** idiom based on logic gates and signal feedback that make self-reference obvious.
2. Truth Tables – a **table** driven look at how inputs become outputs, showing how to combine gates to specify the logical operators that will be needed.
3. Successor Vectors – a **vector** look at truth tables which allows for representing them numerically, a prerequisite for the abstraction of attractor structures.
4. Attractor Structures – a **pictorial** representation which shows fixed points, multi-valued attractors, and phase, all critical for concepts of equality.
5. Non Linear Logic (NLL) – a formal **typographic** system of logic; a symbolic basis for proofs and theorems, which surpasses the limitations of the *mark*. This representation is of potential interest to the logicians.

Each of these tools enable the determination of the total number of logical forms, both linear (combinatorial) and self-referential (sequential). Furthermore, there is a one-to-one correspondence between all five tools. They are different ways of looking at the same problem space. They support each other and inform each other. Most significantly, they agree on the total number of possible logical forms; strong evidence that we are looking at the problem the right way.

Digital Circuits

We'll only be concerned with *schematics* of digital circuits; their actual electrical instantiation will not be needed. However, as a historical aside, early in this research project some effort was expended physically implementing self-referential digital circuits with modern hardware. Within the electrical limits of the technology, the predictions of the formal system to be discussed here were confirmed. Suffice it to say it was a cool period of research while I served as an adjunct professor at Sacramento State University.

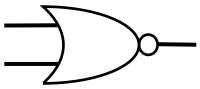
The central idiom in digital circuits is the logic *gate*. It has one or two inputs and just one output. Schematically, a gate is represented as a closed figure. Typically, the inputs are on the left and the output is on the right. Gates are connected to each other by simple lines, representing the electrical connections on a circuit board. An input, but more often an output, may have a small circle appended to the closed figure, which means any signal will be inverted. The notation for the logical values has three typical forms; the logician will use true (T) and false (F), the digital engineer will use 1 and 0, and the electrical engineer will use high (typically +5 volts) and low (typically 0 volts). Just to confuse things even further, the +5 volts may be called the *source* (or, get this, *Vcc*) and the 0 volts will be called the *sink* or *ground*. There is no shortage of standards.

Each line connecting gates (or connecting to the outside world) is treated as a variable and given a label, typically a single character such as 'A' or 'B'. Each gate is also given a name, such as AND or NOR, and is usually highlighted in SMALL CAPS. Each gate will also be given a symbol, such as '^' for AND, or '~' for NOT. Again, there is no shortage of standards, but all these various

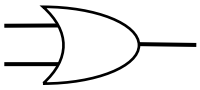
notations allow for very concise, accurate, and error free representation of intent. They become second nature pretty quick.

For those new to all this there are only 16 possible one and two input gates. Surprisingly, there are actually only 4 closed figures used to indicate these 16 possibilities; the differences all come down to where those pesky little inversion circles show up on the closed figures.

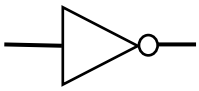
Below is the NOR gate



It is most like the *mark* notation in Laws of Form. Note the small circle on the output to indicated inversion. An OR gate (∨) by contrast, has the same shape, but is missing the circle.



A NOT gate (~) looks like this



which is also like the *mark* of the form, just with a single input rather than the dual input of the NOR gate (or even more).

In a moment, all 16 gates will be presented in a single table. But before committing to that, a preview of the next tool is in order. This is because the next tool comes in two flavors.

Truth Tables

Digital engineers divide digital circuits into two broad categories: *combinatorial* and *sequential*. A combinatorial circuit has no feedback loops, a sequential circuit does. That is the entire difference. Note that the engineer doesn't use the term self-reference, it is buried under taxonomical layers; sequential, feedback loops – like water to a fish.

The truth table for a combinatorial circuit will have *n* input columns (one for each free input), precisely one output column (complicated circuits doing multiple things may identify multiple outputs, but that is an engineering optimization, not a logical distinction), and 2^n rows.

Combinatorial Circuits

The job of the truth table is to map between all the possible inputs and the specific output each permutation of the inputs creates. Here is the truth table for the OR gate.

A	B	OR
F	F	F
F	T	T
T	F	T
T	T	T

This is a formal definition of what the word *or* more or less means in conventional usage, but in this case its either A, or B, or *both*.

For the NOR gate, the logic is just the inverse.

A	B	NOR
F	F	T
F	T	F
T	F	F
T	T	F

Note how the input columns between these two tables are the same, but the output column (column 3) is inverted with respect to the OR gate. If you wish to correlate with the *mark* notation, T is m (the marked state), and F is n (the unmarked state).

The truth table for the NOT gate is even simpler.

A	NOT
F	T
T	F

Based on two-input tables, the output column cell for each row has just two possible values, T or F. There are therefore $2^4 = 16$ possible truth tables with up to two inputs. Thus, we need 16 gates. If you are wondering how the single input gates fold into the dual input gates, you are ahead of the class.

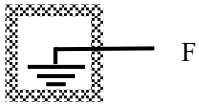
Each logic gate has a name, two actually, one used by engineers, the other used by logicians. (The lack of taxonomical overlap is rather staggering.) In addition, each gate is given a symbol, like in math or algebra. (Here the two fields finally find some common ground.)

This is a lot to take in on a single pass, but it can be made simpler with a single table.

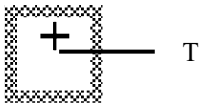
Table of Logic Gates

Below is the table of 16 gates, with gate icons, symbols, gate names, and logical names. The truth tables have been paired up to save space; makes it look like they have two output columns – but it's just a shorthand, one that also clearly shows how the gates pair up by inverses.

A	B	F	T
F	F	F	T
F	T	F	T
T	F	F	T
T	T	F	T

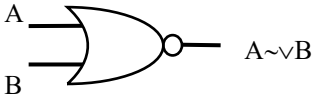


SINK
(Contradiction)

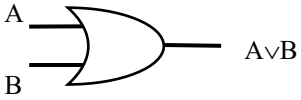


SOURCE
(Tautology)

A	B	A~∨B	A∨B
F	F	T	F
F	T	F	T
T	F	F	T
T	T	F	T

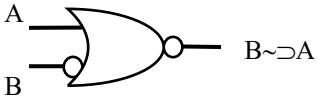


NOR
(Joint Denial)

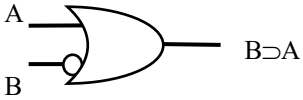


OR
(Disjunction)

A	B	B~⊃A	B⊃A
F	F	F	T
F	T	T	F
T	F	F	T
T	T	F	T

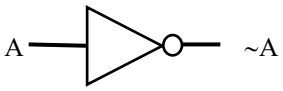


DISIMPLIES
(Not if B then A)

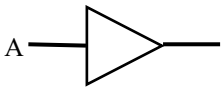


IMPLIES
(If B then A)

A	B	~A	≈A
F	F	T	F
F	T	T	F
T	F	F	T
T	T	F	T

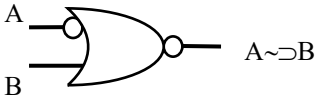


NOT
(Negation)

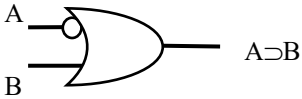


BUF
(Proposition)

A	B	A~⊃B	A⊃B
F	F	F	T
F	T	F	T
T	F	T	F
T	T	F	T

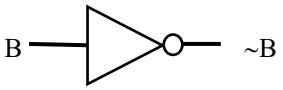


DISIMPLIES
(Not if A then B)

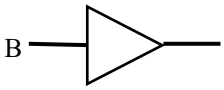


IMPLIES
(If A then B)

A	B	~B	≈B
F	F	T	F
F	T	F	T
T	F	T	F
T	T	F	T

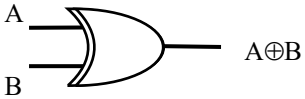


NOT
(Negation)

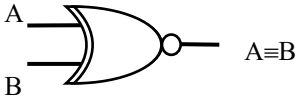


BUF
(Proposition)

A	B	A⊕B	A≡B
F	F	F	T
F	T	T	F
T	F	T	F
T	T	F	T

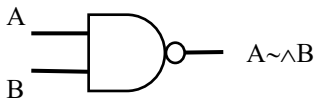


XOR
(Exclusive Or)

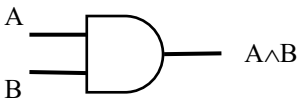


COMP
(Bicomparator)

A	B	A~∧B	A∧B
F	F	T	F
F	T	T	F
T	F	T	F
T	T	F	T



NAND
(Alternative Denial)



AND
(Conjunction)

A few patterns are worth mentioning. Inputs are typically labeled with a single letter, caps in our case. The input columns are all the same. The rows are listed in what you might call binary order, with the 2nd column (B in the above table) changing the fastest. Each gate has a unique pictogram, a digital name (xor) in small caps, a logical name (negation), and a symbol (⊕). Two gates ignore both inputs, these are the SOURCE and SYNC gates (Vcc & ground) and define (by arbitrary convention) true and false. Four gates ignore one input, these are the single input gates of NOT and BUF (buffer). The remaining 10 gates use both inputs, but are rendered with only three outlines, the differences coming down to the locations of the small invert circles.

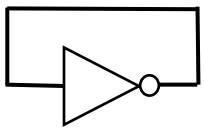
The symbols are pretty straight-forward, except that they depend on character sets which don't translate well between different application programs. To sidestep that issue, several of the operators are indicated with a sequence of symbols. For instance, '∧' indicates the AND gate, so '~∧' indicates the NAND gate. In a compound symbol, the negation sign '~' always precedes the other symbol.

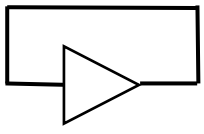
Usually, the inputs are on the left, the output on the right, so reading (and signal propagation, i.e. causation) is left to right.

Sequential Circuits

Sequential circuits confuse inputs with outputs, some outputs come back around (feedback) to be used as inputs. Therefore, such truth tables have more than one output column. In a maximally self-referential circuit, all inputs come from outputs. The truth tables for such forms (circuits) will have the same number of output columns as input columns.

The Liar's Paradox is a sequential circuit. So is the Circular Argument.





Successor Vectors

Successor vectors make sense when a circuit is fully, or at least mostly, self-referential, lots of feedback loops. They are useful for sequential circuits but have minimal utility for combinatorial circuits.

A fully self-referential circuit restores the symmetry between the inputs and the outputs as there are now *n* inputs and *n* outputs; a symmetry that was broken by combinatorial circuits (non-self-referential forms) as they had *n* inputs, but only *1* output.

With the symmetry restored we can now represent all the inputs as a single column of numbers, and of course, also represent all the outputs as a single column of numbers. You may have noticed that in all the truth tables presented so far, the order of the input columns was constrained by a pattern. The right most input column alternated between F and T, the 2nd right most column alternated between 2 F's and 2 T's, the 3rd right most column alternated between 4 F's and 4 T's, etc., etc., etc. This allows us to easily translate each row into a single number, a binary number. To make this clear we shall now present the generalized truth tables with 0's and 1's instead of F's and T's with the natural mapping of F => 0 and T => 1. Remember, no shortage of standards.

We shall also prepend a column *x* which is the decimal representation of the binary number implied by the input columns, and postpend another column *S(x)* which is the decimal representation of the binary number implied by the output columns. It should be obvious by now that we have chosen to list the inputs in numerical order.

Like this

x	A	B	C	A:≈B	B:~C	C:≈A	S(x)
0	0	0	0	0	1	0	2
1	0	0	1	0	0	0	0
2	0	1	0	1	1	0	6
3	0	1	1	1	0	0	4
4	1	0	0	0	1	1	3
5	1	0	1	0	0	1	1
6	1	1	0	1	1	1	7
7	1	1	1	1	0	1	5

Figure 1 – **Generalized Truth Tables Encode a Successor Function**: Each input row is given a number, *x*, treating each input column (A, B, C) as a binary bit. The input rows are presented in numerical order. The output columns are also given a number in the same fashion, *S(x)*.

The way to think about this is to regard the truth table as specifying a successor function. Given an input x, its successor is *S(x)*. Both *x* and *S(x)* are from the same set of numbers; $0 : 2^n-1$, so the second successor of *x* is *S(S(x))*. In other words, a successor function permits self-reference; the output can be fed back into the input; the domain and range are the same. Such functions are referred to as *iterators*.

There are lots of possible successor functions. Some are easy to compute, some difficult. In a computer, it can be more efficient to represent the difficult to compute ones as a table, in this case

a one-dimensional array. In this abstraction, the input is just the location of the n^{th} element, and the output is the value of array at that location (starting at zero). It is this abstraction that will be referred to as a successor vector.

Like this.

2	0
0	1
6	2
4	3
3	4
1	5
7	6
5	7

Figure 2 – **Successor Vector**: The index into the array is the input value, x, the value in the array is the output value, S_x. The index is zero based.

As pedagogy, we have chosen an example of a successor vector where each possible number occurs only once. This is a special case. In such cases, there will be a reverse successor vector; one which maps each successor back to its one and only predecessor. Such successor vectors, if large and difficult to compute without secret information, can form the basis for a public key cryptographic system.

How many such systems are there? A little notation will help here. Let lowercase n be the number of variables, and uppercase N be the number of rows in the truth table. Thus $N = 2^n$.

One has N choices for the first element, then any of the other numbers except that one for the second choice, so that's $N-1$, then any for the 3rd element, except the first two, so that's $N-2$. Obviously, the number of reversible forms is $N!$. Or just for completeness, $(2^n)!$.

However, that is not all the possibilities. If the predecessors are not unique, then duplicate successors occur. Allowing duplicate successors means that every element, all N of then, can take on any of the N values, which is just N^N. The formula for the total number of logical forms is (2^(2^n))^n; 2 choices (T, F) for each of the 2^n rows, for each of the n output columns.

This is super-exponential growth.

Attractor Structures

The successor vector concept invites one to take the successor of the successor; just repeatedly use each successor as the input to the next one – simple iteration. This generates a growing sequence of numbers. It should be obvious that eventually this repetitive process will produce a number seen before, and then the sequence will repeat, forever.

These repeated sequences can be represented as simple graphs – in this case graphs from the theory of networks from computer science; a specialty called graph theory. The graphs that result

from successor vectors are a subset of general graphs. In graph theory there are nodes and edges; edges connect nodes and may be unidirectional or bi-directional. In this case, all edges are unidirectional, and every node has only one successor, but may have multiple predecessors, or even none.

These graphs can be represented pictorially. Each node gets a unique number, from 0 to N-1. These graphs are called *attractor structures*, a name borrowed from chaos theory where dynamical systems tend to orbit around small regions called *attractors* in some abstract phase space.

For our example we'll use the successor vector from Figure 2 but will also present the digital circuit schematic that encodes the same self-referential form.

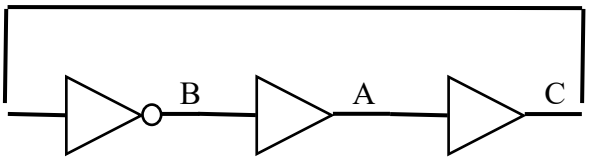


Figure 3 – **Successor Vector**: The index into the array is the input value, x, the value in the array is the output value, S_x. The index is zero based. Below the successor vector is the digital circuit schematic for the same self-referential form.

The successor of 0 is 2, its successor is 6, then 7, 5, 1 and finally back to 0. Elements (nodes) 3 & 4 are successors of each other. So, we have two attractors, two pure cycles without any stems, in our attractor structure: one with 6 nodes the other with 2.

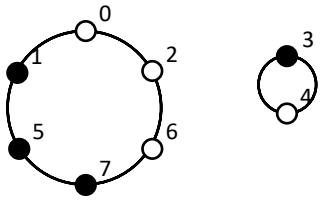


Figure 4 – **Attractor Structure**: A single attractor structure consisting of 2 attractors, each a pure cycle (no stems, i.e., all successors have but a single predecessor). One has 6 nodes, the other 2. Convention is to traverse clockwise. The last column determines if a node is white (F) or black (T).

Conventions

Figure 4 shows the two attractors in the attractor structure. There are several conventions in effect here. First, is that traversal is clockwise, thus 2 is the successor to 0, rather than 0 being the successor to 2 (1st attractor). Second, the lowest number node is usually presented at the top of the attractor. Third, white represents that the bit represented by the last column is false, while black

indicates that it is true. When there is good reason to violate these conventions, the text will explicitly call it out.

In this particular example, all successors have but a single predecessor, thus all the attractors are pure cycles, no stems. This means this form is reversible; there will be one *and only one* other self-referential form that runs these two cycles in the other direction. Note that if all the attractors are singlets or doublets, then the same form is its own inverse.

Significance

There are three observations that are significant.

First, there is more than one attractor. Therefore, this form is multi-valued, in this case there are two possible solutions: attractor 1 or attractor 2.

Second, neither of the possible solutions is a fixed point. This form is neither true nor false, indeed both solutions are paradoxical, but the frequencies of the paradoxes are different. The doublet is the fastest oscillation possible; it oscillates between true and false at the paradox frequency. The 6-node attractor also oscillates, but 3 times slower. Having admitted imaginary truthvalues (the 2-node attractor) we have also opened the door to complex ones.

Not only does this form have multiple solutions, but each solution comes with a phase. And that's the third significant observation. Where have we seen both multiple values and phase? That's right, quantum physics.

Reversibility

As a little bit of practice, let's derive the inverse of this self-referential form. It is a four-step progress; reverse the direction of the attractors, use them to compute the inverse of the successor vector, use it to complete the inverse truth table, and then use it to determine the inverse digital circuit schematic.

Reverse the Attractors

Inverting the attractor structure is easy, just run around each attractor in the counterclockwise direction. Thus

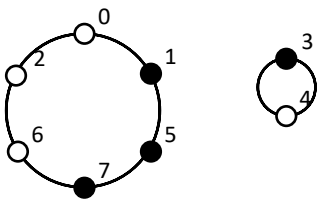


Figure 5 – **Inverse Attractor Structure**: Simply reverse clockwise and counter-clockwise.

Invert the Successor Vector

It is possible to simply exchange the indexes with the elements and resort the array on the index, or you can simply follow the inverted attractors. Either technique will yield the inverted successor vector.

To invert the successor vector by following the inverted attractors, just populate the successor vector from the inverse attractor structures. Thus, the successor of 0 is now 1, then 5, 7, 6, 2, and back to 0. The doublet is its own inverse and so does not change.

1	0
5	1
0	2
4	3
3	4
7	5
2	6
6	7

Figure 6 – **Inverse Successor Vector**: Swap indexes with elements.

Determine the Inverse Truth Table

Simply plug in the inverted successor vector and change the output columns to the binary representation for each node, then determine the gates that produce each output column.

x	A	B	C	A:≈C	B:≈A	C:~B	S(x)
0	0	0	0	0	0	1	1
1	0	0	1	1	0	1	5
2	0	1	0	0	0	0	0
3	0	1	1	1	0	0	4
4	1	0	0	0	1	1	3
5	1	0	1	1	1	1	7
6	1	1	0	0	1	0	2
7	1	1	1	1	1	0	6

Figure 7 – **Invert the Truth Table**: Pull in the inverse successor vector, then change the output columns so the binary representation is correct for each node, then determine the gates for each column.

Determine the Inverse Digital Circuit Schematic

Lay out the circuit. In this case, we took a shortcut and simply flipped each gate.

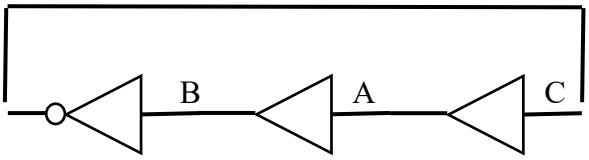


Figure 8 – **Inverse Digital Circuit Schematic**: In this example, each column in the reverse truth table only depended on one input column. What changed is which column and perhaps the gate.

That's interesting. The inverse digital circuit simply flipped each gate around.

Stems

With N nodes, the attractor structure might consist of a single attractor, or up to N pure attractors (in which case they are all singlets). When a node has more than one predecessor, we get *stems*, and thus less than N attractors. All stems eventually lead to the cyclic part of an attractor. It is worth taking a moment to provide a feel for the variations that are possible.

Attractor Structure with Stems

In this example, there are two attractors, a singlet and a doublet. For variable C (bit 0), the singlet is true (real) and the doublet paradoxical (imaginary). This form is thus multi-valued with different roots. Stems can be short, long, branched, hairy, or additional descriptive nouns. A node may have none, 1, or even up to N – 1 stems.

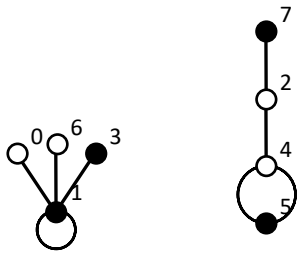


Figure 9 – **A Multi-Valued Attractor Structure**: There are two attractors, one a singlet and one a doublet. The singlet is a fixed point, true for bit 0 and false for bits 1 & 2 (001 = 1). The doublet is true for bit 2, false for bit 1 and paradoxical (imaginary) for bit 0, (100 = 4, 101 = 5).

A Self-Referential Tautology

Here is another attractor structure. This one has been judiciously selected for a couple of reasons. First, it provides another example of stems, just to continue the pedagogy of providing examples that reveal more of the possibilities. Second, this one has interesting properties; it has two fixed points, both singlets, and is thus multi-valued, but with duplicate roots for B & C.

However, it is also tautological (for both B & C). As a further refinement, we've introduced two shades of gray for the nodes where B & C have different real truthvalues.

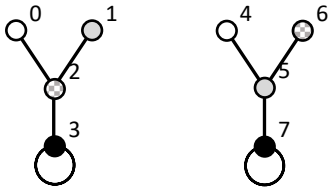


Figure 10 – **A Tautological Attractor Structure with Two Fixed Points**: The first four nodes have node 3 as their eventual successor, as 0 & 1 lead to 2 and eventually to node 3. The last four nodes have the same structure: but with node 7 as their eventual successor. Nodes 4 & 6 lead to 5 and eventually to node 7. Because the two low order bits (variables B & C) are true in both cases (node 3 & node 7) this logical form is tautological for both variables.

Truth Table and Successor Vector

It's easy to create the truth table; just enter the successor vector and represent each node in binary. Then determine the logical gate that produces each output column.

x	A	B	C	A:A	B:A⊃C	C:A∨B	S(x)
0	0	0	0	0	1	0	2
1	0	0	1	0	1	0	2
2	0	1	0	0	1	1	3
3	0	1	1	0	1	1	3
4	1	0	0	1	0	1	5
5	1	0	1	1	1	1	7
6	1	1	0	1	0	1	5
7	1	1	1	1	1	1	7

Figure 11 – **Truth Table for the Tautological Attractor Structure of Figure 10**: Note that A is the free input, but B depends on C and C depends on B; and both depend on A. Thus, this form is self-referential.

Digital Circuit Schematic

Here is the circuit. Note that A is a free input. However, whatever the value of A, either true or false, the eventual outputs B & C are always true, and thus tautological. The stems represent transitory states. The longer a stem, the longer it might take such a form to settle out to stable or repetitive state (one clock cycle per node). Seems time has entered into logic.

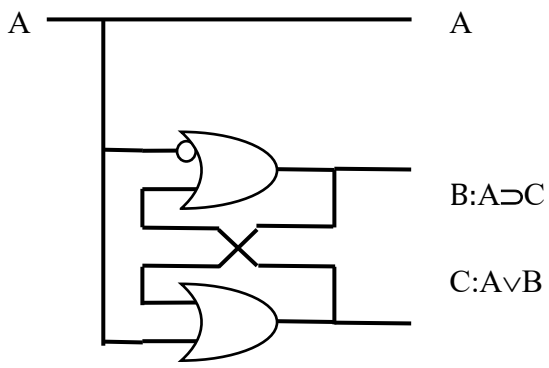


Figure 12 – **A Tautological Self-Referential Digital Circuit**: This form is both self-referential and tautological. Any self-respecting electronic engineer would delete this from the circuit board replacing its B & C outputs with a tap to the source voltage.

One of the interesting things about this logical form is that it is both self-referential and tautological. Because it is self-referential, formal systems of linear logic such as Bertrand Russell's theory of types does not allow it, and thus such systems cannot derive it in a proof, yet it is tautological. Here is a true theorem that conventional logical systems cannot prove to be true, yet it is a simple logical form and obviously, even trivially, true.

In hindsight this is saying something devastating about the field of logic and about the emphasis given to proof 2,500 years ago by Euclid. It is not possible to start with a set of axioms, not matter how clever, complete, or erudite, then utilize any finite set of rules to step nimbly from true proposition to true proposition and reach every possible logical form that is true. Self-reference subverts this very basic expectation.

No wonder the logicians find self-reference to be a nightmare.

Formal Systems

We have covered digital circuits, truth tables, successor vectors, and attractor structures. Four tools down, one to go. Now we turn to the formalism beloved of logicians, a typographic formal system. It is austere.

Historical Setting

The gold standard in human reasoning for the last 2,500 years has been the axiomatic method, a formalized style of reasoning first championed by the Greek Euclid. In its modern incarnation, a finite set of symbols is chosen, both operators and operands, a set of base values, and a set of hopefully both complete yet minimal axioms, and finally a set of rules of transformations. These are called *formal systems*. Usually, such systems are typographic, which pretty much just means that every expression in the formal system can indicated by a one-dimensional string of symbols. There will be a grammar that defines when a string is well formed or not, the not well-formed

strings are regarded as meaningless, and the well-formed strings should be either true, false, or contingent.

The design objective is to have the smallest, clearest set of primitives (axioms) that permit one to derive each and every true statement without error. These are the principles of *correctness* and *completeness*. These derivations are called proofs, a sequence of expressions where each is the result of a legal transformation, and the final typographic expression is the theorem that has just been proven.

At the start of the 20th century it looked like this holy grail might be achieved, the ambition being to subsume all of both logic and mathematics under the paradigm of formal systems. In 1900 the brilliant mathematician David Hilbert laid out a program of 10 key problems that needed to be solved to achieve this lofty goal. In the course of tackling these problems, the paradoxes started showing up, first in set theory, then just about everywhere else, and by 1931, with the publication of Gödel's Incompleteness theorem, the situation had become dire. At stake was the supremacy of human reasoning as the ultimate expression of rationality in the universe. Paradise barred.

Self-Reference

All of the limit theories that arose from this effort have one thing in common; the subversion comes via self-reference. Early efforts at a resolution focused on banning self-reference, the best known of which was Bertrand Russel's theory of types, but it seems that self-reference is canny; it always finds a way to sneak in. Perhaps, self-reference is not a blight on logic, but absolutely fundamental to its true (if I even dare use that word) nature.

So, the approach taken here is to embrace self-reference, to embed it into a formal system of logic at the begging so it forms a key part of the foundations, in the hope of preventing its re-entering the fray unexpectedly, uncontrollably, and often enough devastatingly.

A Self-Referential Typographical Formal System of Non-Linear Logic

In the course of presenting the previous four tools (digital circuits, truth tables, successor vectors, and attractor structures) we have been leaving subtle clues as to how we might formalize a typographic non-linear logic; a logic that allows self-reference to be explicitly represented and evaluated with symbolic expressions.

First, we have used symbols for logical variables (the operands), letters, typically capitalized (such as, A, B, C, etc.), but we'll also allow lowercase and might as well allow words as well. Second, we've been using a set of symbols for the logical operators, (specifically ~≈∨⊃∧⊕≡) and to keep the symbol set concise, have allowed the NOT operator (~) to be conjoined with the OR, IMPLIES, and AND operators. (Yes, we might have done the same with the XOR operator (⊕) rather than a separate symbol for comparison (≡), but equality is such a fundamental concept that we broke the pattern here. I know, I know, no shortage of standards.) So far, this is all pretty conventional.

However, there was one new symbol, and with it a new concept, that has been flying below the radar. It's called the label operator (:) the lowly colon. It means assign the symbol on the left

to represent the expression on the right. Think of it as an *iteration* operator. It has also implied that the right-hand expression is clocked. This is a very restrictive constraint, with the potential to limit the generality of the resulting system. It intruded into our awareness via the paradigm of digital circuits. Let us now enlarge that paradigm.

Concurrency

Figures 10 and 11 show a logical form with both linear and self-referential elements. It was chosen because it highlighted some more variations of stems on attractor structures, as well as because it was both multi-valued and tautological. It had three variables, labeled A, B, & C, each assigned to represent an expression, specifically the logical result of the expression to the right of the colon. Let us thus take the headings of all three output columns of the truth table for that entire logical form, and place them into a single typographic string, separated by commas, like this:

$$\mathbf{A:}A, \mathbf{B:}A\supset C, \mathbf{C:}A\vee B \tag{1}$$

Since B and C each depend upon each other, it is clearly self-referential. The intent of the comma operator is to indicate what evaluations must be *concurrent*; in this case given initial values for A, B, & C, new values are computed concurrently. In this way there is no explicit clocking, no absolute time, just the relativity of computing each new generation of the full set of values, one generation after another. The successor vector provides a static look at the form, the attractor structures a dynamic look.

This is a very democratic representation, but usually, we want to focus on only one of the outputs. Whichever one it is, it is the one which we wish to know and to preserve in a proof. So, time to pick another standard, again. In the attractor structures, we used a binary evaluation to determine a number for each node and settled on black nodes implying the last bit (variable C in this case) to be true and a white node to imply the last bit was false. It made the last variable the one most readily read off the attractors, but the first variable is more consistent with the typical reading of expressions from left to right. Having the last variable be the important one feels like burying the message, so let's rewrite equation (1) this way

$$\mathbf{C:}A\vee B, \mathbf{A:}A, \mathbf{B:}A\supset C \tag{2}$$

or given that A is a free variable, even better this way

$$\mathbf{C:}A\vee B, \mathbf{B:}A\supset C \tag{3}$$

Given another self-referential form with three variables, one with C tautological, such as this expression

$$\mathbf{C:}A\supset B, \mathbf{B:}C\supset A \tag{4}$$

(in which B is not tautological) we might seek a proof that would convert the first form into the second in a finite number of steps. If you are skeptical that C is tautological in both Equations (3) and (4), Figure 13 shows the truth table (with successor vector), attractor structure, and the digital circuit schematic.

x	A	B	C	A	B:C⊃A	C:A⊃B	S(x)
0	0	0	0	0	1	1	3
1	0	0	1	0	0	1	1
2	0	1	0	0	1	1	3
3	0	1	1	0	0	1	1
4	1	0	0	1	1	0	6
5	1	0	1	1	1	0	6
6	1	1	0	1	1	1	7
7	1	1	1	1	1	1	7

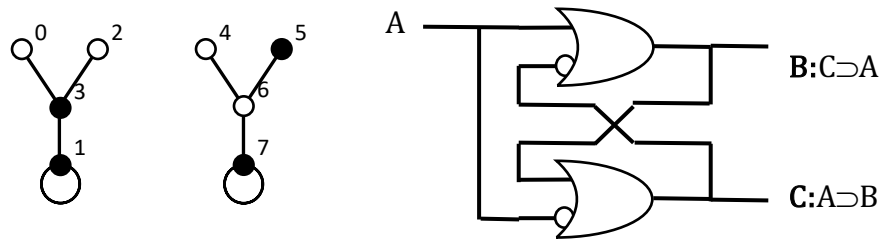


Figure 13 – **Another SR form Tautological for C**: Truth table (with successor vector), attractor structure, and digital circuit for the form of Equation (4) which like the form in Equation (3) is tautological for C.

The fact that there are two attractors in the attractor structure reflect the fact that A is a free variable. Indeed, the nodes group themselves according to the value of A. This is a special case, so one should not expect that free variables determine the number of attractors.

Ganged Gates

In the examples above, the right had expression has needed only one gate because the variable on the left was only dependent upon two variables. Introducing 3-input gates, or even worse, n-input gates gets quickly out of control. Fortunately, we can use 2-input gates in various combinations to achieve any n-input gate. Before the label operator (:), when individual gates were interpreted to be clocked this could not have worked. But relaxing the clocked constraint to the more general concept of concurrency allows the use of unclocked 2-input gates to define whatever Boolean logic is needed.

In general, the combination of 2-input gates to specify a particular n-input gate is not unique. The permutations are well understood from current Boolean logic as these combinations are all

combinatorial. In other words, the right-hand side of a label expression must not contain any self-reference. This may appear to be a step backwards, but in practice presents no problem.

An open question is whether there is a logical form that can be regarded as simplest, similar to the concept of reduction to lowest terms for fractions. Minimal circuit depth might play a role here, particularly for practical applications.

To be clear the set of gates to the right of the colon operator are regarded as implementing a ganged gate, a higher order truth table (N>2). This allows a finite set of 2 input gates to be the primitive algebra of logic. A set of ganged gates are clocked as unit, not individually.

It is useful to be ultra clear here. On the rising edge, every input to a set of ganged gates is sampled. Before the clock's falling edge, each ganged circuit must compute its output, so that on the falling edge, every output will be transmitted to all the downstream inputs. The time between rising edge and falling edge must be long enough for the circuit to compute the logical value without error, and the time between falling edge and rising edge must be long enough for the signals to propagate to the downstream inputs.

We will return to this implicit invocation of time in a later chapter.

Self-Referential Forms

Single Variable Self-Referential Forms

With this abstraction, we can now evaluate the **Liar's Paradox** as both a digital circuit schematic and as a sequential truth table:

x	A	A:~A	S(x)
0	0	1	1
1	1	0	0

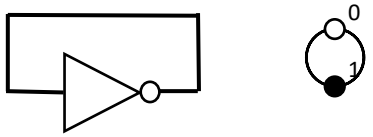


Figure 14 – **Liar's Paradox**: Here are two ways to represent the Liar's Paradox, as a truth table where all inputs come from outputs, and as a digital circuit schematic consisting of a single NOT gate feeding back on itself. The NOT gate should be regarded as clocked.

And likewise, the **Circular Argument**:

x	A	A:≈A	S(x)
0	0	1	0
1	1	0	1

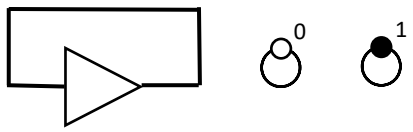


Figure 15 – **Circular Argument**: Here are two ways to represent the Circular Argument, as a truth table where all inputs come from outputs, and as a digital circuit schematic consisting of a single BUF gate feeding back on itself. The BUF gate should be regarded as clocked.

There are no other single input self-referential forms. There is, however, the truth table with all F's (corresponds to ground, or *sink*) and the one with all T's (corresponds to +, Vcc or *source*). Two rows, two possible values for the one output column, thus a total of $2^2 = 4$ possible single variable truth tables.

Dual Variable Self-Referential Forms

With these tools we can now tackle more difficult self-referential forms, such as this two-stage self-consistent paradox:

x	A	B	A:~B	B:~A	S(x)
0	0	0	1	1	3
1	0	1	0	1	1
2	1	0	1	0	2
3	1	1	0	0	0

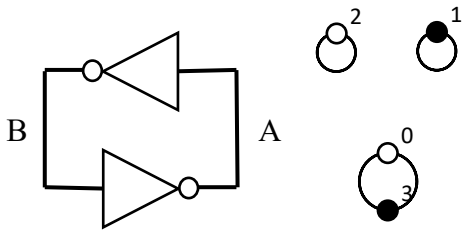


Figure 16 – **A Two-Stage Self-Consistent Paradox**: Two NOT gates feeding back onto each other appear at first glance to be self-consistent, a simple indeterminacy between true and false (both solve the system, ala both are fixed points), but the truth table reveals a hidden solution, a paradoxical solution, that echoes with the thesis that imaginary truthvalues are a viable logical concept. As before the clocked gate abstraction is in effect.

Note that rows 2 & 3 are self-consistent, but rows 1 and 4 are contradictory. Hmmm… A two-stage circular argument where each stage is a negation, is generally regarded as having two stable solutions, two fixed points. True is consistent, so is false. But here we seem to have a third solution, one where imaginary truthvalues might have some merit. It seems this form can also be paradoxical, oscillating between true and false on each tick of the clock.

There are four rows for dual variable truth tables; each can have one of two possible values, T or F, but there are also 2 columns; thus, there are $2^4*2^4 = 256$ possible such truth tables.

N-Variable Self-Referential Forms

Just as an example, we'll consider a 3-variable self-referential form with some unspecified 3 input gates. Thus:

A	B	C	A:f(A,B,C)	B:g(A,B,C)	C:h(A,B,C)
F	F	F	F/T	F/T	F/T
F	F	T	F/T	F/T	F/T
F	T	F	F/T	F/T	F/T
F	T	T	F/T	F/T	F/T
T	F	F	F/T	F/T	F/T
T	F	T	F/T	F/T	F/T
T	T	F	F/T	F/T	F/T
T	T	T	F/T	F/T	F/T

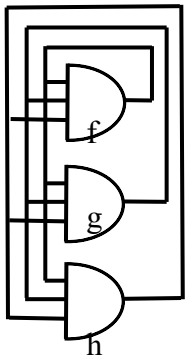


Figure 17 – **Generic 3-Input Self-Referential Form**: Three unspecified gates (f, g, h), 3 variables (A, B, C), 8 rows. Each gate depends on all of the others.

While the gates are unspecified, it should be obvious that all the permutations of all the 8-row, 3-column truth tables cover all the possibilities. Thus, there are $2^8*2^8*2^8 = 256^3 = 16{,}777{,}216$, possible such truth tables.

These numbers get big, really, really fast. What they mean is that far and away most logical forms are self-referential, yet our conventional logical systems duck them. Indeed, Russell's theory of types was designed to prevent them. From this perspective, we know very little about logic. This is a tad bit humbling, or perhaps terrifying has the more appropriate emotional context. We regard math and logic as the epitome of human reason, the very foundation of rational thought, the "queen of the sciences." Can we really be this far behind? Are we this naïve about logic?

Just to raise the stakes;

n	(2^(2^n))^n	(2^n)(2^n)
1	4	4
2	256	256
3	16,777,216	16,777,216
4	$1.84x10^{19}$	$1.84x10^{19}$
5	$1.46x10^{48}$	$1.46x10^{48}$
6	$3.94x10^{115}$	$3.94x10^{115}$
7	$5.28x10^{269}$	$5.28x10^{269}$

Figure 18 – **Scaling Laws**: For n variables, the scaling law for truth tables (2^{nd} column) is the same as for successor vectors (3^{rd} column).

Wow. My calculator fails at 8 variables. This is the same scaling law for both successor vectors and truth tables.

Epilog

Something to keep in mind. This chapter makes an assumption – it assumes iteration is well behaved. Engineers know where this limit is, and do not cross it.