

# ICONIC AND SYMBOLIC CONTAINMENT IN LAWS OF FORM

William Bricken

December 2009

## Overview

At the turn of the twentieth century, the mathematical community adopted a radical plan to put mathematics on a firm foundation. The idea was symbolic formalization, the representation of concepts using encoded symbols that bear no resemblance to what they mean. At the same time, C.S. Peirce developed Existential Graphs, an iconic representation of logic that is drawn rather than written [ref]. Iconic forms are images that look like what they mean. In 1967, G. Spencer Brown reintroduced Peirce's iconic system in a more general algebraic style as Laws of Form [ref]. Today, mathematics remains symbolic, while other communication media have evolved into visual and interactive experiences. This essay explores the possibility of an iconic mathematics.

Laws of Form (LoF) is an algebraic system expressed in an iconic notation. We interpret LoF as a calculus of containment relations, and develop intuitive, symbolic, and iconic descriptions of the Contains relation. The representation of the LoF calculus is analyzed from both the iconic and the symbolic formal perspectives, with a focus on the descriptive structures necessary to align the two representational techniques. An iconic notation resembles its interpretation, which is at variance with modern mathematical conventions that strictly separate syntax from semantics. Iconic form is displayed in two and three dimensions, while symbolic form unfolds as encoded one-dimensional strings of tokens. Iconic variables stand in place of arbitrary patterns of zero, one, or many objects, while symbolic variables stand in place of single expressions. Symbols conform to speech, while icons conform to vision.

The iconic mathematics of LoF provides a semantic model for both a relational and a functional calculus. The operation of putting an object into a container forms an algebraic quasigroup. When further constrained by LoF rules, the functional calculus provides a new perspective on Boolean algebra, an equally expressive calculus that is both non-associative and non-commutative. LoF can be interpreted as propositional logic; iconic containment as well provides a new, non-symbolic deductive method.

We show four varieties of iconic notation to illustrate the interdependence of representation and meaning. Iconic formal systems extend the ability of symbolic notations to express parallelism, transformation across nesting, structure sharing, and void-based transformation. The expressive neutrality of

several symbolic conventions (labeling, grouping, arity, null objects, variables) is examined in light of the LoF iconic calculus. Computational examples in both symbolic and iconic notations are compared side-by-side. We show that inducing symbolic representation on essentially spatial form has led to widespread publication of erroneous information.

# CONTENTS

- 1 Introduction
  - 1.1 Hilbert's Program
  - 1.2 The Dimension of Representation
    - 1.2.1 The Cost of Symbolic Abstraction
    - 1.2.2 A Symbolic Boolean Example
    - 1.2.3 Iconic Formal Systems
  - 1.3 Laws of Form
    - 1.3.1 The Arithmetic of Distinction
    - 1.3.2 Incommensurability
- 2 Containment
  - 2.1 Containment Intuitively
  - 2.2 Formal Containment
  - 2.3 The Contains Relation
    - 2.3.1 A Possible Definition
    - 2.3.2 Multiple Contents
  - 2.4 The Contains Relation Reformulated
- 3 Notation
  - 3.1 Parens Notation
  - 3.2 Annotated Parens Notation
  - 3.3 Predicate Calculus
  - 3.4 Ordered Pairs
  - 3.5 PUT Functions
  - 3.6 Directed Acyclic Graphs
  - 3.7 Variety of Notations
- 4 Model Constraints
  - 4.1 Irreflexive and Asymmetric
  - 4.2 Physicality
  - 4.3 Transitivity and Intransitivity
  - 4.4 Special Objects
    - 4.4.1 The Empty Container
    - 4.4.2 The Universal Container
  - 4.5 Arrangements
    - 4.5.1 Inductive Definition
    - 4.5.2 Construction of Arrangements
    - 4.5.3 Nested Containers
  - 4.6 Containment Revisited
- 5 Iconic Algebra
  - 5.1 The Computational Algebra
  - 5.2 Equations
  - 5.3 Transformation Rules

- 5.4 The Description of Transformation
  - 5.4.1 Exactly One
  - 5.4.2 Nothing Else
  - 5.4.3 Context-Free Rules
  - 5.4.4 Incidental Structure
- 6 The Role of Variables
  - 6.1 LoF Pattern-Variables
  - 6.2 Variables That Partition
  - 6.3 Rule Schemas and Set-Variables
  - 6.4 Principle of Relevance
- 7 Computation
  - 7.1 Iconic and Symbolic Calculus
    - 7.1.1 Initials of the Arithmetic
    - 7.1.2 Initials of the Algebra
    - 7.1.3 Computational Initials of the Algebra
  - 7.2 Properties of the Iconic System
    - 7.2.1 General Recursive Template
    - 7.2.2 Articulated Definition
    - 7.2.3 Arithmetic
    - 7.2.4 Algebra
    - 7.2.5 Interpretation
    - 7.2.6 Container Semantics
  - 7.3 Rule Application by Pattern-Matching
  - 7.4 Single Variable Calculus
- 8 Abstract Algebra [CONTAINS SOME MINOR ERRORS]
  - 8.1 The PUT Function
  - 8.2 Construction of Arrangements
    - 8.2.1 Functional Construction
    - 8.2.2 Notational Flexibility
      - 8.2.2.1 Commutativity
      - 8.2.2.2 Associativity
  - 8.3 The PUT Magma
    - 8.3.1 Not Associative, Not Commutative
    - 8.3.2 Non-associative Cancellation
    - 8.3.3 The Problem of Labeling
    - 8.3.4 Non-associative Division
  - 8.4 Properties of the PUT Function
  - 8.5 Finite Set Theory and LISP Programming
  - 8.6 Functional LoF
    - 8.6.1 Functional Arithmetic
    - 8.6.2 Functional Algebra
    - 8.6.3 Compositional Properties
    - 8.6.4 LoF as PUT Operations

- 9 Remaining Modeling Issues
  - 9.1 Inherent Parallelism
  - 9.2 Deep Transformation Rules
    - 9.2.1 Deep Pervasion
    - 9.2.2 Deep Replication
    - 9.2.3 Permeability
  - 9.3 Structure Sharing
    - 9.3.1 Structure Sharing Transformation
    - 9.3.2 Inconsistency of Reference
  - 9.4 Void-based Computation
  - 9.5 Iconic Languages
    - 9.5.1 Stacked Blocks
    - 9.5.2 Paths
      - 9.5.2.1 Iconic Rules for Path Notation
      - 9.5.2.2 Perspective
- 10 Logic and Boolean Algebra
  - 10.1 Deconstructing Logic
  - 10.2 Interpretation as NOR
  - 10.3 Logic as PUT Functions
    - 10.3.1 The Logical Quasigroup
    - 10.3.2 Quasigroup Deduction
  - 10.4 Mappings to Boolean Algebra
    - 10.4.1 Banaschewski
    - 10.4.2 Kohout and Pinkava
    - 10.4.3 Schwartz
    - 10.4.4 Cull and Frank
    - 10.4.5 Meguire
    - 10.4.6 Not Boolean Algebra
  - 10.5 Asymmetric Duality
  - 10.6 Cognitive Interpretations
- 11 Conclusion
  - 11.1 A Theory of Representation
  - 11.2 Laws of Form
- 12 References
- APPENDIX Examples of LoF Reduction via Pattern-Matching
  - A.1 Pattern-Matching in the Arithmetic
  - A.2 Use of the Pattern-Variable
  - A.3 Partitioning by Pattern-Variables
  - A.4 Proof of Modus Ponens
    - A.4.1 Cognitive Interpretations
  - A.5 Parallel Proof of Modus Ponens

- A.6 Proof of the Subsumption Theorem
- A.7 A Logic Puzzle
  - A.7.1 Cognitive Interpretations
- A.8 Use of Virtual Arrangements

## LIST OF FIGURES AND TABLES

Figure 1.1: Examples of Iconic Notation for Arrangements of Containers  
Figure 2.1: A Variety of Physical Containers  
Figure 3.1: Relationships Between Representations  
Figure 8.1: Commutativity -- Flexibility of Notation  
Figure 8.2: Associativity -- Flexibility of Notation  
Figure 9.1: Arithmetic Initials Expressed as Stacked Blocks  
Figure 9.2: Algebra Initials Expressed as Stacked Blocks  
Figure 9.3: Computational Initials Expressed as Stacked Blocks  
Figure 9.4: Arithmetic Initials Expressed as Paths  
Figure 9.5: Algebra Initials Expressed as Paths  
Figure 9.6: Computational Initials Expressed as Paths  
Figure 10.1: The Boolean Hypercube with Parens Vertices  
Figure 10.2: The Boolean Hypercube Labeled with PUT Compositions  
Figure 10.3: Equivalence Classes of Parens Arrangements

Table 1.1: Some Representational Losses  
Table 3.1: Annotated Parens Notation and Relational Formulas  
Table 3.2: LoF Notation, Relational Notation and Graph Notation  
Table 3.3: Six Notations for a Specific Container  
Table 3.4: An Example Arrangement in Six Different Notations  
Table 4.1: Notational Varieties for an Empty Container  
Table 4.2: Construction of Containment Arrangements One Formula at a Time  
Table 4.3: Construction of Arrangements One Function Application at a Time  
Table 4.4: Arrangements and Conjunction of Relational Formulas  
Table 4.5: Symbolic Constraints that Define the Meaning of Containment  
Table 5.1: The Theory of Equality  
Table 6.1: Constants and Variables in Annotated Parens Notation  
Table 6.2: The Involution Rule Schema  
Table 6.3: Deconstructing Annotated Parens Notation  
Table 7.1: The Iconic Transformation Rules of LoF  
Table 7.2: An Example of Rule Application  
Table 7.3: LoF Single Variable Calculus  
Table 8.1: Notational Inducement of Properties  
Table 8.2: PUT, LISP and Sets  
Table 8.3: LoF Rules as Properties of PUT  
Table 9.1: Steps to Convert Parallel Notation into Sequential Notation  
Table 10.1: The Map from Logic to Parens and to Ordered Pairs  
Table 10.2: Generalized NOR as an Interpretation of Containment  
Table 10.3: LoF Arithmetic as NOR Logic  
Table 10.5: Boolean Algebra as PUT Functions  
Table 10.6: Functional and Iconic Deductive Efficiency  
Table 10.7: Banaschewski's Map  
Table 10.8: Catalan and Rooted Tree Sequences  
Table A.1: Parens and Ordered Pair Notations for the LoF Transformation Rules

# 1 Introduction

Modern technology has initiated many changes in how knowledge is represented and acquired. One trend is toward visual rather than textual forms of expression. A similar trend is occurring within mathematical notation. Feynman diagrams, fractals, knot theory, string theory, diagrammatic reasoning, category theory, silicon circuit design, each of these fields relies upon the spatial representation of mathematical ideas. Like geometry, these fields can be seen as addressing inherently spatial concepts.

The American logician C.S. Peirce was the first to contrast symbolic, indexical, and iconic notations, within his development of the field of semiotics at the turn of the twentieth century. He was also the first to conceptualize iconic logic, the subject of this essay, as his Existential Graphs. The insight that logic can be displayed spatially was virtually lost as mathematicians and logicians alike followed Hilbert and Russell into a mathematical notation based solely on encoded symbols.

In this essay, we begin to explore the reconceptualization of symbolic formal systems as iconic formal systems by examining a single mathematical concept expressed in both symbolic and iconic languages. In 1967, G. Spencer Brown published *Laws of Form* (LoF), which presents an iconic algebra that is quite similar to Peirce's Existential Graphs. LoF addresses a single semantic concept that Spencer Brown calls distinction. We interpret the concept of distinction geometrically as enclosure, that is, as complete spatial containment. A generic name for the formal study of iconic enclosures is boundary mathematics.

The icon of containment is a two-dimensional enclosure that distinguishes its inside from its outside. Characteristically, iconic notation represents containment in a form that also illustrates containment. Figure 1.1 shows several examples of the iconic representation of LoF containment arrangements.

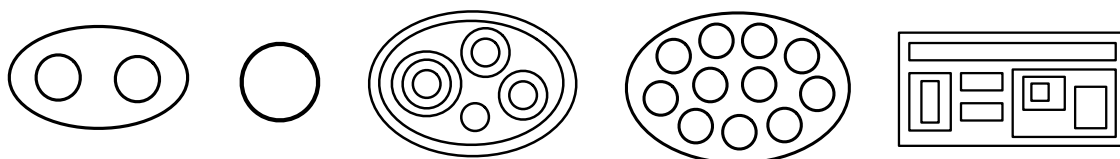


Figure 1.1: Examples of Iconic Notation for Arrangements of Containers

We annotate LoF forms with labels to develop a relational description of arrangements of containers. This permits the iconic axioms of Spencer Brown's system to be represented in symbolic predicate calculus, and supports comparison of the structural and interpretative differences between the two approaches to notation. Although the LoF calculus can be interpreted as propositional logic,



like Spencer Brown we approach it as an algebra, in particular as the algebra of a single binary relation, Contains. The Contains relation can also be expressed as a function, PUT. To PUT Object-a into Object-b means that Object-b Contains Object-a. PUT exhibits the algebraic structure of a quasigroup.

An algebra can also be used as a computational system. Equations identify valid substitutions, and substitution can be applied to identify the equivalence classes formed by the algebra. The arithmetic of LoF, for example, establishes two equivalence classes over the iconic forms that constitute the language of LoF. When LoF icons are interpreted as containers, the calculus identifies valid transformations of arrangements of containers. This container calculus conforms to our intuitive notions of how containers can be arranged. Surprisingly, it also describes logical inference. Under a logical interpretation, the container calculus is a new non-symbolic deductive method.

## 1.1 Hilbert's Program

About one hundred years ago, well prior to the advent of digital display technologies, David Hilbert exerted his considerable influence to complete the exclusion of diagrams and spatial intuition from formal mathematics. For over two millennia, the paradigm of mathematical rigor was Euclid's treatise on geometry, Elements. Literally every trained mathematician studied Euclid. Symbolic notation began to appear in Europe in the sixteenth century; Euclid's Elements was clearly a non-symbolic text. It did however incorporate logic, those formal principles of reasoning that lead from given premises to valid conclusions.

In the mid-nineteenth century the discovery of non-Euclidean systems of geometry contributed to a loss of trust in human mathematical intuition, particularly spatial, visual intuition.

"...for two centuries mathematics has had harsh words to say about visual evidence. The French mathematicians around the time of Lagrange got rid of visual arguments in favor of the purely verbal-logical (analytic) arguments that they thought more secure." [45, p 30]

"[A diagram] is dispensable as a proof-theoretic device...proof is a syntactic object consisting only of sentences arranged in a finite and inspectable array." [46, p###]

The endeavor to place all of mathematics on a firm foundation lead to competing schools of thought, dominant among those were logicism and formalism. Logicism posited that a foundation could be found purely in logic. Logic separates content from process, providing sufficiently general guidelines for reasoning

that it applies to any content. Logic is also firmly rooted in language, so that symbolic strings provide a natural mode of expressing the ideas of logic. The logical style is to assert statements, called sentences, that are either TRUE or FALSE. The truth of a sequence of reasoning steps rests upon the truth of the beginning premises, together with a set of constraints (the rules of logic) that assure that truth is maintained.

Formalism takes the next logical step, to strip symbolic strings of all meaning, and to consider their structure and manipulation to be independent of any actual circumstance.

"A theory, a rule, a definition, or the like is to be called formal when no reference is made in it either to the meaning of the symbols (for example, the words) or to the sense of the expressions (e.g. the sentences), but simply and solely to the kinds and orders of the symbols from which the expressions are constructed." (Carnap 1937 ;cf. 1942:232; 1943:6)

The elimination of meaning from representation freed formalists to assume that how a concept is written does not impact what the concept means. This in turn permitted the assumption that the iconic notations of Peirce, Frege, Venn, and even Euclid were at best only syntactic variants that contributed nothing to understanding, and at worst were sources of error and confusion.

"...for a particular set of theories which encode some of our highest standards for correct reasoning -- the theories of expression and proof which are operative in geometry and technical logic -- the representations which are currently sanctioned are uniformly sentential. This tradition is so pervasive that it is rarely explicitly commented upon...the overall sentential style is typically presented as if there were no sensible alternative."  
[greaves p2]

Hilbert's formal agenda has been rigorously pursued throughout the twentieth century, so that today

"Mathematics is a game played according to simple rules with meaningless marks on paper." (David Hilbert, c. 1900)

The success of an entirely symbolic approach, followed closely by the rise of the use of string-processing techniques in digital computers, has led to the expression of mathematics almost exclusively in symbolic string languages.

"...despite the obvious importance of visual images in human cognitive activities, visual representation remains a second-class citizen in both the theory and practice of mathematics." [15, p3]

## 1.2 The Dimension of Communication

Types of communication media are differentiated by the dimensions each requires for display. Words are one-dimensional strings; photographs are two-dimensional images; films require two dimensions of space and one dimension of time. As the dimension of representation is reduced, rules for interpreting the placement, arrangement and grouping of objects become increasingly necessary.

One-dimensional forms of expression (writing, speech, music, mathematical symbolism) are sequential media. There is so little space in one-dimension that it is necessary to spread display out over a temporal dimension as well. Symbolic representation is a one-dimensional temporal encoding technique that arose from the spoken word. The strength of speech and of writing is that meaning is encoded in a sequence of sounds and tokens, the individual sounds and tokens themselves being meaningless. This strategy is very flexible, new concepts can be communicated by arbitrary new sequences of sounds and tokens. Symbols excel in describing abstract concepts such as freedom and perfection. The weakness of symbolic expression is that it takes a lot of cognitive effort both to encode and decode. Codes range from the binary system through human languages. Modern children dedicate a dozen years to learning the symbolic techniques of reading, writing, and arithmetic.

Human languages, of course, are not solely symbolic. Early languages, such as Egyptian hieroglyphics and Mayan glyphs, are iconic, as are many modern Asian languages. Iconic languages are not limited in abstraction nor in clarity, suggesting that an iconic mathematics would also not find inherent limitations in the expression of abstract formal concepts.

Two-dimensional forms of expression (drawing, vision, painting, diagrams) are parallel media. Iconic representation utilizes placement in space, which provides sufficient expressibility to escape temporal encoding altogether. Iconic forms can embody meaning itself, by expressing physical reality with sufficient realism as to be taken for that reality. The strength of drawing and of painting is that vision is engaged effortlessly. Iconic expression excels in depicting concrete appearance, it is weak in telling stories and showing dynamics. Spreading two-dimensional forms over time gives us movies, television and computer interfaces to address those weaknesses.

Each form of communication media provides a different type of experience. The structural limitations of each media constrain what can be communicated through that media. The different dimensional media convey different meanings and experiences, as illustrated in the sequence,

"house"



*Actual house* →

The word "house" is encoded to elicit rather than to present an image of a house. The encoded two-dimensional cartoon image of a house has few realistic features, but it is sufficient to draw upon memory immediately with little cognitive effort. The artistic rendering of a house adds depth, context, and idealism. The three-dimensional model permits a more thorough exploration of the space defined by the house, and the realistic photograph provides an image of an existent house with definitive characteristics. An actual house, rich in experience, can only be suggested, it is outside of the page's space of representation.

Houses are containers of sorts. They delineate a space together with many other features to provide creature comfort. The shell of the house suggests a more abstract concept, enclosure. How is this enclosure represented by the various dimensional media? The first step is to remove all that is characteristic of an object such as a house, to consider only the concept of containment.

Boxes, for example, might represent containers, and arrangements of boxes (such as the rooms in a house), arrangements of containers. A containment arrangement, in actual circumstance, might be a collection of boxes inside other boxes. Iconic representation illustrates the arrangement by highlighting certain abstract properties, such as boundaries and containment relations. Below, an actual arrangement of boxes is reduced to a binary string by successive steps of abstraction. All of these representations maintain the relationship of containment that exists between the original physical objects.

actual      3D      2D      (( ))      110100

The actual arrangement of containers is first reduced to an image. Although the image is printed on a flat page, it retains the sense of three dimensionality of the actual containers. The image does restrict our ability to move around the containers in space. The realistic image is abstracted to a three-dimensional outline of boundaries and containment relations, with some aspects of the object boundaries obscured by the singular viewing perspective of the image. The 3D abstract outline is reduced to two dimensions by tracing the outlines of the top edges of the containers. In two dimensions, the relative locations of the actual objects is still maintained, but relations in depth are lost. The 2D containment arrangement can be reduced to one dimension by representing each container by two delimiting tokens. Delimiters continue to maintain the structure of containment but the ability to localize containers in space is lost. As well, the delimiter configuration imposes a false structure on the

containers, suggesting that they are arranged in a row. The delimiter arrangement is itself a symbolic code, perhaps made clearer by transcription into binary ones and zeros.

The sequence of dimensional reduction,

actual -> 3D image -> 3D outline -> 2D outline -> 1D delimiters -> 1D string

shows that the abstract concept of containment can be expressed across dimensions, within each dimensional media. Almost all of the properties of the physical objects are lost, but the elected essential property of containment is maintained in the iconic spatial structure of the representation. This sequence of abstractions maintains a visual correspondence between actual circumstance and abstract representation. The final binary digit representation, however, requires knowledge of the encoding procedure, losing all immediate visual recognition of the original relations.

### 1.2.1 The Cost of Symbolic Abstraction

Predicate calculus expresses iconic form as an abstract one-dimensional symbolic strings of tokens.

$(( ))$                        $(( )_2 ( )_3)_1$                       1 Contains 2    AND    1 Contains 3

To arrive at a symbolic form, each actual container is labeled, and a relational label, Contains, is created to symbolize the act of containment. Object labels are combined with the relation label to generate the symbolic string. The symbolic string is also assigned a meaning. The labels 1, 2, and 3 mean (or refer to) the actual physical objects. The label Contains means that some object (the first argument to the relation) actually contains some other object (the second argument to the relation). When there is more than one containment relationship, symbolic logic is also required. The word AND serves to combine the multiple relationships into a whole, the symbolic arrangement.

Symbolic representation flattens the nesting relation between objects, thereby removing any spatial correspondence between actual and symbolic. The symbolic form requires interpretation, in effect a reconstruction of the actual containment arrangement represented by the iconic forms. That is, the iconic images, regardless of their dimension, provide an illustration of the meaning of the symbolic form. Iconic form incorporates both representation and meaning; symbolic form is purely representation, it acquires meaning via reference to an actual containment arrangement.

Several issues arise when iconic form is converted into symbolic form, even when both are expressed in one spatial dimension. In addition to labeling, for

example, the nature and meaning of empty space changes. These issues can be summarized by the observation that a representation is inherently biased by the restrictions of its display media. The sequence of abstractions that convert actual circumstance to images and to symbols loses information at every step. The final step, symbolic encoding, loses all information that may have been accessible by sense perceptions.

Each type of media biases both what can be communicated and how communication can be achieved. The intent of symbolic mathematics is to remove all of these biases, so that no room remains for loss of rigor or for confusion in interpretation. Unfortunately, this symbolic idealism fails to heed McLuhan's central thesis, that regardless of type, the medium is the message. Symbolic representation impacts meaning just as much as any other representation, although in its own particular way.

Table 1.1 shows some of the representational losses that occur when translating across static dimensional media. These losses are cumulative.

TRANSFORMATION	LOSSES
actual --> 3D image	tactile, haptic, aural, spatial choice
3D image -> 2D outline	depth, perspective
2D outline -> 1D delimiter	spatial arrangement
1D delimiter -> symbolic	void, depth, meaning

Table 1.1: Some Representational Losses

Symbolic notation is representational minimalism. It converts all sensory input into cognitive effort. Presumably, abstract thought is sufficient to restore that which is lost. Mathematical symbolism explicitly does not care about these losses, the Platonic ideal is that the content of mathematics is not within the grasp of sensory information anyway. Our concern is with symbolic structure that has achieved the status of mathematical semantics, independent of actual intended meaning. The thesis is that the symbolic constructions of labeling and sequencing and grouping are artifacts of symbolic representation. These constructions have been embraced so dearly by mathematical communication over the last century that their status has been elevated from artifacts of a dimensional media to definitional properties of mathematics itself.

Symbolic notation is rife with embedded meaning. The mathematical concepts of variables, arity, associativity, and commutativity have lead to deep structure

and to deep understanding, particularly in function theory. But mathematics can be much more, it can also address human senses directly. An example is the incipient field of mathematical and scientific visualization. To reincorporate the human sensorium, it will be necessary to generalize mathematical notation to include iconic and dynamic formalisms. Iconic notation also incorporates embedded meaning, but the iconic concepts are quite different than the symbolic concepts. Iconic representation calls upon the senses for interpretation, symbolic concepts do not.

One of the costs of converting mathematics to pure symbolism is that teaching and learning elementary mathematics turns into memorization, removed from concrete manipulation and from visceral experience. Young students learn that mathematics is not connected to the common sense of their bodies. Research in childhood cognitive development has shown that very young infants demonstrate a relatively sophisticated understanding of containment, and that toddlers learn the meaning of the word “in” prior to learning other spatial concepts such as on, under, in front of, and behind [ref]. Physical containment, then, might provide an experiential foundation upon which students can build their understanding of abstract concepts.

### 1.2.2 A Symbolic Boolean Example

Symbolic notation incorporates many conventions. We propose that some of the structural characteristics of symbolic representation have been absorbed into the semantics of mathematical notation, confounding meaning with representation. For example, the logical connective AND is defined as binary, commutative, and associative. Are these properties syntactic or semantic? Do they describe the rules for recording the concept (and thus would vary with notation), or do they describe what AND means?

The traditional definition of Boolean operators is mired in antiquated systems of communication, usage, and notation. In particular, the recent advent of both formal diagrammatic reasoning and parallel computational paradigms raises comparative questions about the meaning and utility of the common symbolic representation of logic. It is common practice in computational languages and in silicon hardware design to treat AND as though it connects an arbitrary number of arguments. Such connectives are variary, they apply to zero, one, or many arguments. Commutativity and associativity are not relevant concepts for a variary operator.

The connective AND does not necessarily have to be binary. The meaning of “x AND y” is very similar to the meaning of “x AND y AND z”. In both cases the conjunction is TRUE only when each argument is TRUE. By extension, a single argument AND is TRUE when that argument is TRUE. A zero argument AND would be TRUE should the non-existent argument be TRUE. Here however it is better to

avoid vacuous truth, and define non-existence as FALSE. It is in the treatment of non-existence (i.e. void) that symbolic and iconic notations diverge the most. There are strong mathematical reasons to consider AND as a binary Boolean connective, but we suggest that there are equally as compelling reasons to define AND to apply to any number of arguments. For example, in written and spoken language AND is variary. We write  $x$ ,  $y$ , and  $z$ .

In many computing languages, the order of occurrence of the arguments to AND is of critical importance, violating the logical definition of conjunction as a commutative operation. In parallel computational environments, grouping and sequencing of arguments loses meaning, all arguments are considered concurrently.

Does an iconic notation that represents conjunction in a form that is parallel and variary redefine conjunction? Do non-standard notations incorporate group theoretic properties implicitly? The answers depend upon whether or not mathematics is seen to be Platonic (i.e. eternal, unchanging, in a virtual reality of its own), or evolutionary. This contrast bears a resemblance to Lakatos' thesis that mathematical concepts evolve through a negotiated process of exclusion, reconceptualization, and generalization [ref]. Lakoff [ref] presents the thesis that mathematics is an essential human endeavor that is deeply rooted in the human physical experience. Today, we appear to be exiting an exclusionary phase in which iconic and visceral mathematical forms had been deemed to be insufficiently formal. Freeing mathematical notation of its adherence to string representations not only provides new tools and techniques, it broadens our conceptualization of mathematics itself.

### 1.2.3 Iconic Formal Systems

An iconic formal system represents abstract concepts through illustration rather than through symbolic abstraction, while at the same time maintaining the formal rigor of a mathematics. Iconic formalisms have been extensively studied over the last few decades, after being out of favor for a century. During the last half of the nineteenth century, while mathematics itself was being formalized, iconic and symbolic representations grew hand-in-hand. Frege and Peirce, for example, both used spatial notations to make foundational contributions to the ideas associated with formal systems [refs]. Hilbert's program to cast mathematics solely in terms of symbol manipulation essentially banished iconic notation from mathematics [ref]. Russell and Whitehead's *Principia Mathematica* (1910) eschews all graphs, tables, and diagrams in favor of purely textual, symbolic forms [ref]. The symbolic formalization of mathematics in the twentieth century is a fundamental departure from a mathematics rooted in several millennia of human experience and intuition. We are left with a



mathematical system of symbols that intentionally separates representation from meaning.

In contrast, iconic systems do not maintain a strict separation of meaning and representation. A picture, for example, looks like what it identifies. LoF's spatial distinction also illustrates what it means. Capturing the strength of illustration is very difficult for purely symbolic systems. It takes a plethora of symbols to describe the idea of spatial containment, an idea that is illustrated directly by an abstract image of an enclosure.

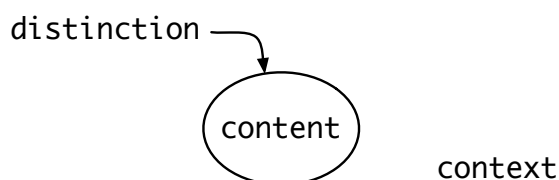


### 1.3 Laws of Form

Laws of Form is a formal calculus of arrangements of spatial enclosures that Spencer Brown calls distinctions. The two equations of the arithmetic specify valid transformations of arrangements of containers. Since iconic representation rests on fundamentally different formal concepts than symbolic representation, interpreting LoF in conventional terms can be significantly misleading.

#### 1.3.1 The Arithmetic of Distinction

In LoF, Spencer Brown presents a two-dimensional spatial representation of his seminal concept of distinction. This representation separates the plane being drawn upon into two distinct areas, inside of and outside of the enclosing mark of distinction. The spatial enclosure can be interpreted as a container that distinguishes its contents from its context.



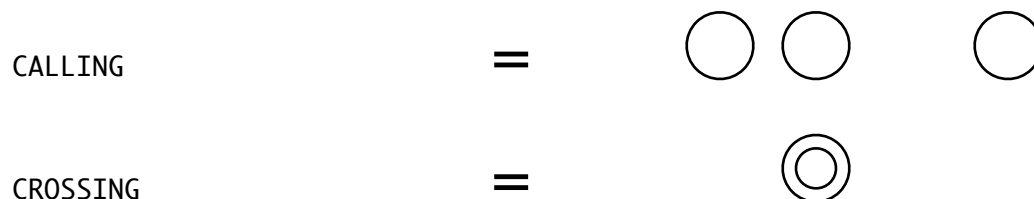
LoF is often considered enigmatic. For example, Spencer Brown defines distinction as "perfect continence". We take this to mean enclosure without overlap, that is, complete containment. An enclosure is spatial in that it occupies two-dimensions on the page; it is iconic because the notation looks like what it means.

Containers can be composed in two distinct ways, by placing a second container inside the original and by placing a second container outside the original. We

interpret placement on the inside as an illustration of the concept of containment, and placement on the outside as an illustration of non-containment.



Spencer Brown describes the development of his iconic calculus from the beginning, taking distinction as the single core concept. He constructs two rules for the arithmetic of distinctions, one rule for reducing each of the two primary ways of composing containers.



The declaration of equality between forms of containment constructs an algebra. These equations are taken to be axiomatic, their meaning as transformations of containment arrangements is not interpreted within the calculus. The structure of these iconic equations is reminiscent of common formal concepts. Calling, for example, might be read as a rule of idempotency, and Crossing as composition with an inverse. As another example, Calling may be interpreted for logic as the equation TRUE OR TRUE = TRUE, and Crossing as NOT TRUE = FALSE.

Some formal properties of LoF's iconic notation are so visually obvious that they can easily lead to inaccurate transcription of LoF into conventional symbolic notation. The conversion from iconic to symbolic form highlights some of the difficulties inherent in formalizing iconic calculi within a sequential notation, and in generalizing sequential notation to describe parallel computational processes.

In constructing the calculus of distinctions, Spencer Brown avoids reference to almost all conventional mathematical constructs, such as sets, functions, logic, mapping, arity, and cardinality. By not incorporating concepts taken to be foundational to mathematical systems, Spencer Brown proposes a remarkable idea: a formal system that does not incorporate sets or logic or counting can provide an alternative foundation for mathematics. In particular, Spencer Brown proposes an iconic foundation.

### 1.3.2 Incommensurability

As is typical of mathematical systems, the symbolic logic that has been developed over the last century is composed of tightly interlocking concepts. Symbolic names are associated with actual objects in a domain of discourse. Variables stand in place of these objects. The quality (i.e. correctness, expressibility, consistency, soundness, completeness, decidability) of a symbolic system is calibrated in reference to string-based structures, logics, and meanings. The different style of representation employed by iconic formal systems requires a different conceptual approach to much of what is taken to define symbolic formality. The proofs, mechanisms, and hard-gained knowledge of symbolic formal systems to some extent simply do not apply to iconic formalisms. For example, existentially quantified symbolic variables refer to at least one object. In contrast, iconic variables identify patterns rather than objects. A pattern can be the absence of an object, a single object, or an arbitrary group of objects. Pattern-variables do not require the concept of quantification.

The theorems and assertions that define our knowledge of symbolic form must be carefully evaluated when applied to iconic form. This essay begins with both a symbolic representation and an iconic representation of containment, and then finds a connection between the two. But because the symbolic approach is well-developed, it has little flexibility in reaching toward iconic techniques. In comparison, reconstructing iconic form to be symbolic is largely a matter of degradation, of removing the essential features of the iconic notation until both the representation and the intention are no longer visual or intuitive.

The aspects of representation that are incommensurable between symbolic and iconic notations are

- the incorporation of logic and sets, i.e. what is foundational
- the use of void, i.e. the meaning of nothing at all
- the use of variables, i.e. how to identify arbitrary structure
- the use of computation, i.e. how to manipulate forms

As an example, within an iconic notation, equality does not necessarily invoke the concept of a truth value. The two iconic forms might be said to be indistinguishable rather than equal in value. Spencer Brown suggests an interpretation of the equals sign as "is confused with". Iconic equality may assert that certain visual properties are to be ignored. The following two equations can have very different intentions.

$$A \ B = B \ A \qquad A \ B = \begin{matrix} B \\ A \end{matrix}$$

The symbolic equation on the left can be taken to assert a property of space as an operator, that juxtaposition in space is commutative. The iconic equation on the right can be taken to declare that spatial position does not matter, that

space is not an operator at all. Instead, the relative locations of the forms sharing space are irrelevant, analogous to, for example, the irrelevance of font size to the representation of symbolic forms. The equals sign asserts a structural transformation that does not impact value in the symbolic case; it is a sign of structural identity in the iconic case.

String representation includes an inescapable underlying metric, each character occupies one string position. String position is sequential, so that the relations `ToTheLeftOf` and `ToTheRightOf` are built into the notation. Commutativity is a necessary transformation if only in recognition of the sequencing bias of the string representation. Iconic representation, in contrast, has a dimensional bias (forms lay on the same plane), but not a positional bias.

As another example, the LoF rule of Calling might be written in both symbolic and iconic notations,

$$0 \ 0 \ = \ 0$$



The symbolic equation might mean that the spatial juxtaposition of an object with itself has the same value as the object taken by itself. The equation asserts that symbolic space, as an operator, is idempotent. The iconic equation may be taken to mean that replication of the object is illusory.

Metaphorically, the left side of the iconic equation is what we would see when crossing our eyes, the right side is the actual singular image. Because icons are located within a non-operational space, illusory form can be equated with absence, with empty space. The positional dictate of strings does not permit illusion as a symbolic concept, yet within the iconic representation illusion can be strictly formal: a form is void-equivalent only when it has no impact on the meaning of the space it occupies.

As a final example, the iconic equation of Crossing cannot even be written in symbolic form without introducing symbolic artifacts such as labels for objects, labels for empty space, bifurcation of container tokens into left and right parts, and logical connectives that permit compound expressions. The one-dimensional structure of a string permits juxtaposition but not nesting. To record the concept of nesting symbolically, a delimiting pair of string-tokens, which themselves represent a two-dimensional enclosure, must be constructed. String notation does include tokens of enclosure, but they are largely relegated to incidental notation, grouping tools to (ironically) keep the string notation unambiguous.

In the sequel, we compare iconic and symbolic calculi, demonstrating some of the difficulties of expressing spatial concepts in conventional symbolic notation. Modeling decisions are motivated by the desire to describe containment without introducing additional relational and structural concepts. To keep comparison

crisp, we want to represent LoF in a conventional notation with the least amount of incidental structure. We also require that both iconic and symbolic languages provide a computational model.

## 2 Containment

Intuitive and formal definitions of containment are described, in preparation for a pattern-matching implementation of the LoF equations as transformation rules on both iconic and symbolic representations.

### 2.1 Containment Intuitively

A primary intuitive distinction is between physical containment and the abstract idea of containment represented by a symbolic or an iconic notation. Physical containers possess characteristics of size and shape, so that some containers will fit inside others (Russian dolls, for example), while some will not (the box of apples may not fit into the shopping cart). To address the abstract concept rather than a physical instantiation of that concept, we'll suppress any metric associated with the size or shape of a container.



Figure 2.1: A Variety of Physical Containers

Figure 2.1 shows a variety of physical containers, extruded so that the objects can be seen clearly, but potentially nested in many different ways. For simplicity, containers will be the only type of object. We assume that the meaning of containment can be accurately represented in both symbolic and iconic notations. The simplest container is an empty container. Arrangements of containers are created by placing containers inside other containers. Intuitively, what then are the characteristics of these arrangements?

Without an underlying metric or size, any container can contain any other container. However a container cannot contain itself, it is not possible to put a box inside itself.

A container can contain any collection of other containers. It's possible for a container to contain nothing, or to contain one other object, or to contain many other objects. Contained objects may themselves have contents. The capacity of containers is finite, but assumed to be sufficient for any purpose.

Two containers cannot mutually contain each other. We want to avoid arrangements of containers that are not physically possible. Overlapping and intersecting containers are also excluded, in order to focus on the idea of complete containment.

These restrictions, together with the intuitive image of an enclosure, define valid container arrangements. Since the focus is on containment, all arrangements should have an outer container. Two containers side-by-side, without an enclosing outer container exemplify non-containment rather than containment.

## 2.2 Formal Containment

Containment is a fundamental spatial concept. There are various specialized applications for which containment is defined, such as a sentence that contains a word, or a message that contains some information, or a set that contains all the members of another set. Although the concept is used widely, formal definition is rare. Succinctly, the Contains relation is a finite directed acyclic graph (Section 3.5), while arrangements of unlabeled containers are rooted trees.

Modern mathematics relies on set theory as a basis for description of mathematical ideas. Set theory is a natural place to look first for formal definition, where containment is called the inclusion, or subset, relation. The curly braces that represent sets are even suggestive of containers. An inclusion relation exists whenever every member of a given set is a member of another set. The given set is a subset of the greater set. Strict inclusion forbids the two sets to be equal. Inclusion is defined as a partial ordering; it is reflexive, antisymmetric and transitive. However, containment does not mean constituency, nor does its intuitive description necessarily imply a partial ordering.

The reason that inclusion is not containment is that inclusion does not address nested containers. When we say, for example, that dogs are animals and that animals are living things, we conclude that dogs are living things because the subset relation is transitive. The set of dogs and the set of living things both consist of discrete, comparable objects. However, when we say that the dog ate the bone, and the dinosaur ate the dog, we do not conclude that the dinosaur ate the bone, even when the bone is inside the dog. Bones and dogs may be at the same level of nesting, but bones inside dogs are not. We might say that the

dinosaur indirectly ate the bone, in recognition that being eaten is the direct relationship. Set inclusion is transitive because it does not incorporate these indirect relationships, the idea of being nested at deeper levels that is central to containment structures.

Containment as a concept is also addressed by the field of mereology, the study of part-whole relations. This field does not rely upon set theory but does rely upon the logical structure of predicate calculus. We also develop a description of LoF based on predicate calculus. Doing so, however, changes many of the essential features of the LoF calculus of images. Our objective is to understand the differences between these two formal approaches.

## 2.3 The Contains Relation

We begin by defining containment in the conventional notations of predicate calculus and set theory. We construct a set of labels that stand in place of conceptual container-objects, and define a binary relation  $S$ , so that  $S[1,2]$  reads: Object-1 Contains Contents-2. The domain of the relation (the first argument) consists of individual containers, and the range (the second argument) consists of the contents of the domain object. Container-objects in the range may themselves have contents.

This Contains relation is better called ShallowContains since containment here refers only to those objects directly contained by the domain container. We will use the word Contains to mean ShallowContains.

### 2.3.1 A Possible Definition

In the symbolic approach of predicate calculus, all containers can and must be labeled. We use numerals to identify specific containers, and numerals proceeded by an asterisk to identify the contents of a container. Small letters from the beginning of the alphabet will be used as variables to identify arbitrary containers. A possible relational notation would be,

$$\exists a \exists *a \ S[a,*a] \quad =\text{def} \quad \text{Container-}a \ \text{ShallowContains} \ \text{Contents-} *a$$

This reads: There is an arbitrary container named  $a$  with contents named  $*a$ .

Any particular symbolic occurrence of the Contains relation is called an atomic, or simple, formula. A formula is a statement about containment that is either TRUE or FALSE. The universe of all possible relational formulas is the Cartesian product of the two sets of labels, all possible ordered pairs. The Contains relation is the subset of the set of all possible pairs that conforms

to the definitional constraints. The constraints identify whether or not a formula is TRUE. As a first step, then, here is a possible formal definition:

Let  $C$  be a finite, non-empty set of containers, labeled  $\{1,2,3,\dots\}$ .  
Let  $*C$  be a finite set of sets of containers, labeled  $\{*1,*2,*3,\dots\}$ .  
The Contains relation  $S$  is on  $C$  to  $*C$ , and is a subset of  $C \times *C$ .

It is not necessary that the set of containers be infinite, just that there be enough. The contents of each container are collected together into a set. The collection of contents of all containers,  $\{*1,*2,*3,\dots\}$ , is a set of sets. Valid members of the set of ordered pairs,  $C \times *C$ , are determined by formal constraints that convey the intentions of the intuitive definition. We will call any valid configuration of containers and contents, an arrangement.

The above definition asserts a first constraint on the Contains relation, that any contents are the contents of some container. This Containment Constraint allows the label of a container to be used as an indirect reference to what it contains. Contents can be identified without being named or displayed by identifying their container. The asterisk in front of container labels is said to dereference the label so that it represents the contents of the named container. This approach is analogous to the set theoretic choice between extrinsic definition (explicitly naming the members of a set), and intrinsic definition (naming the property shared by all members of a set). Here, the intrinsic property is the communality of being contained by the same object.

### 2.3.2 Multiple Contents

The contents of any container, as described intuitively, can consist of no objects, of one other object, or of many other objects, with the exception that a container cannot contain itself directly or indirectly.

The symbolic representation of a collection of items requires some relation, or function, or set, that collects the items together. If we have two objects, for example, a description of the two together might be expressed as the set  $\{1,2\}$ , or as the logical conjunction  $1 \text{ AND } 2$ , or as a relation, say  $\text{TakenTogether}[1,2]$ . Although an intuitive visualization may convey the impression that two objects can simply coexist, it is not possible to refer symbolically to the two objects without some sort of connective that relates the two. Symbolically, in this case, we would still write  $\text{CoExists}[1,2]$ .

This grouping requirement comes from the definition of valid expressions in predicate calculus. In particular, an expression must be a rooted tree, so that every expression can be taken to be a single (possibly compound) object. A consequence is that if the Contains relation is to be from containers to contents, then a new type of object must actually be constructed, the content-



object. In the prior definition, arrangements were constructed so that they can be referenced by some single outer container, and sets were constructed as content-objects to collect the contents of any container.

Our goal is to avoid introducing other types of objects into the description of containment. Content-objects are not necessary if the connective role of logical conjunction is permitted to stand in place of the connective role of sets. A possible pattern-matching alternative is to construct a different type of variable (called a pattern-variable) that can be associated with multiple objects. These different representational techniques provide different formal models for the Contains relation. In the case of Object-1 having both Object-2 and Object-3 as its contents, we have these descriptive choices:

Multiple Contents:      1 Contains {2,3}  
                             1 Contains TakenTogether[2,3]  
                             1 Contains 2    AND    1 Contains 3  
                             1 Contains x, and x is bound to both 2 and 3

Since Object-2 and Object-3 are the contents of the same container, the act of containment itself should be a sufficient grouping mechanism; containment is also a grouping operation. The earlier use of the label of Object-1 to indirectly label the contents of that object, \*1, presumes that containment is also grouping. This intermediate labeling, however, does not eliminate the need for a set-object:

Multiple Contents:      1 Contains \*1    AND    \*1 = {2,3}

Can the Contains relation be formalized without an incidental grouping mechanism? More generally, is it possible to express in conventional notation what is visually apparent in the LoF iconic notation?

Multiple Containment:    (2 3)

## 2.4 The Contains Relation Reformulated

We'll express multiple containment via logical conjunction to generate a predicate calculus description of the Contains relation. To do this however, the formal definition of Contains needs to be from containers to containers, not from containers to contents.

Let  $C$  be a finite, non-empty set of containers, labeled  $\{1,2,3,\dots\}$ .  
The Contains relation  $S$  is on  $C$  to  $C$ , and is a subset of  $C \times C$ .

This definition is a relation on a set, mapping objects as containers to the same set of objects being contained. It achieves the objective of avoiding

incidental structure by restricting the objects in both domain and range to the single set of container-objects. This in turn will lead to further modifications. In particular, two special objects must be introduced, an outermost universal container, and an innermost empty contents token.

In the sequel, the Contains relation  $S[a,b]$  refers only to a relation between two containers. We abandon the concept of contents as a separate set-object, and refer instead to the specific individual objects contained in each container. The idea of an arrangement of containers will be formalized later (Section 4.5), for now we take an arrangement to be the intuitive concept of a container and its contents.

Arrangement  $(2\ 3)_1$ :  $S[1,2] \ \& \ S[1,3]$

### 3 Notation

To express containment in both iconic and symbolic languages, we will call upon several different notations.

- Parens notation is a convenient typographical shorthand for two dimensional enclosures. It represents arrangements of LoF distinctions.
- Annotated parens notation is a bridge to a representation of containment arrangements in predicate calculus.
- The predicate calculus description consists of quantified conjunctions of relational formulas. Only one relation other than equality is used, the Contains relation.
- Sets of ordered pairs are derived directly from the predicate calculus description. Membership in the set of valid relational pairs replaces both quantification and conjunction in logic.
- The PUT function provides a functional representation of the Contains relation.
- Directed acyclic graphs are a spatial notation that provides an alternative visualization of both parens and relational notations. Containment graphs model the single binary relation, Contains.

Figure 3.1 presents a summary of the relationships between these representations.

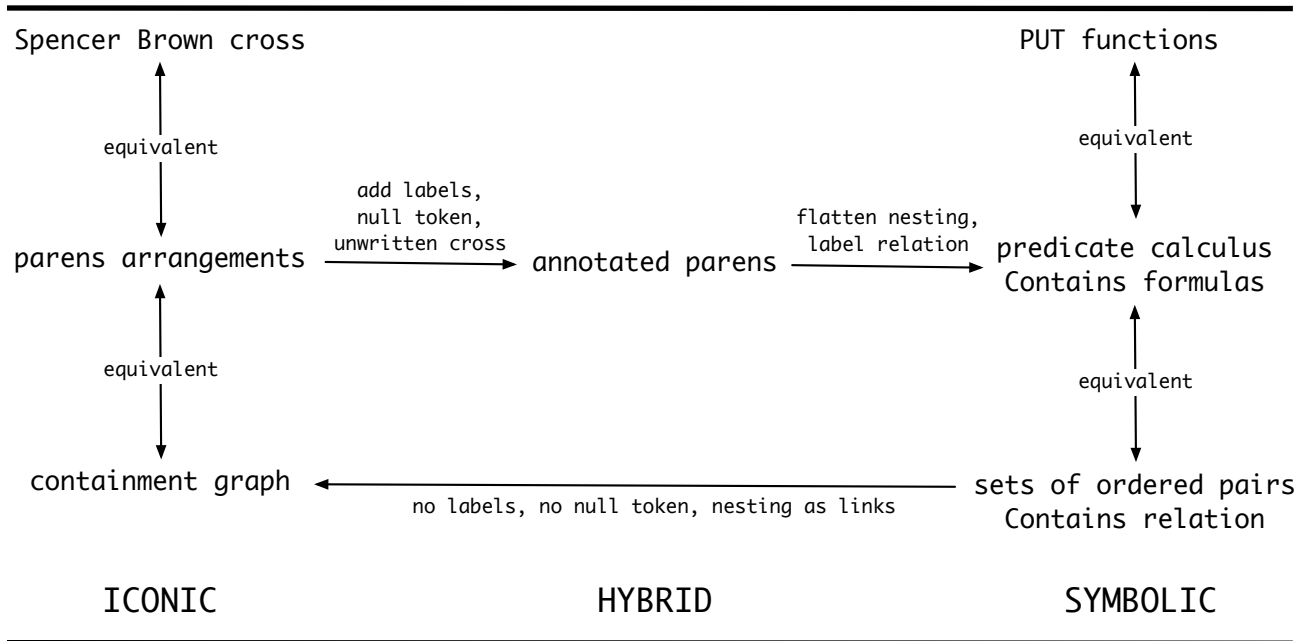


Figure 3.1: Relationships Between Representations

The iconic notations are equivalent, as are the symbolic notations. In converting between iconic and symbolic, labels for containers and for empty space are added. The hybrid annotated parens notation combines aspects of both iconic and symbolic representation; the spatial arrangement of delimiting tokens remains iconic, the identification of containment structures with labels is symbolic.

### 3.1 Parens Notation

Textual delimiters (parentheses, brackets, braces, etc.) permit a typographic representation of containment. We'll use arrangements of parentheses, called parens notation, to visually illustrate containment relations. The two-dimensional enclosure on the Euclidean plane,  $\bigcirc$ , the LoF cross,  $\top$ , and the parens,  $( )$ , all share the same meaning, they represent complete enclosure.

Parens stand in place of themselves, they do not need to be labeled. Parens also enforce non-overlapping containment, it takes the introduction of labels to express the idea that an object can be in more than one container.

Parens notation serves as a visually explicit representation, in contrast to the arbitrary tokens that constitute a symbolic representation. The actual circumstances being described are patterns of containment. Parens arrangements are an iconic representation of these containment patterns. The relationship

between parens and actual circumstance is more of a tracing than a mapping. Actual objects map to individual parens, actual nestings map to individual parens nestings. Parens impose incidental structure on spatial arrangements by putting the objects in a line.

In converting to a symbolic form, a primary objective is to employ parens patterns as a universe of discourse that supports axiomatization in predicate calculus. A complementary primary objective is to employ parens patterns not as an interpretation, but as a calculus. These joint objectives blur the distinction between representation and meaning.

We will take parens arrangements to be the abstract reality that is described by symbols. It is common in Computer Science to define the semantics of one symbol system in terms of another abstract system. To map the iconic notation of containment onto a symbolic notation, we will need to introduce labeling, a universal container, and a null token.

Due to its string-orientation, the formal theory of parenthesis structures treats the left-parenthesis and the right-parenthesis as separate tokens. This theory makes assertions such as, "There must be an equal number of left- and right-parenthesis." The iconic theory of enclosure does not divide its container-objects into two parts, and is thus not comparable to the symbolic theory of parenthesis forms.

### 3.2 Annotated Parens Notation

Annotated parens notation combines both the symbolic and the iconic perspectives. It is intended as a bridge to the symbolic notation of relations. The closing parenthesis of each parens container is annotated with a subscript that identifies it symbolically. Numerals are used as concrete labels and small letters from the beginning of the alphabet as variables standing in place of arbitrary containers. This hybrid notation maintains the iconic structure of containment, and also identifies that structure with symbolic labels.

Here are three versions of an abstract containment formula, going from iconic on the left, through the annotated hybrid in the middle, to purely symbolic on the right. Labels refer to arbitrary containers.

	PARENS	ANNOTATED	RELATIONAL TERMS
Empty Container	( )	( $\emptyset$ ) <sub>a</sub>	S[a, $\emptyset$ ]
Single Containment	(b)	(b) <sub>a</sub>	S[a,b]
Multiple Containment	(b c d)	(b c d) <sub>a</sub>	S[a,b] & S[a,c] & S[a,d]

Table 3.1 shows the complete collection of annotations to parens notation required to convert LoF into a predicate calculus notation. The rationale for these annotations is discussed in the sequel.

TYPE	ANNOTATED PARENS	RELATIONAL TERMS
Specific container objects	$(2\ 3\ 4)_1$	$S[1,2] \ \& \ S[1,3] \ \& \ S[1,4]$
Variable container objects	$(b\ c\ d)_a$	$S[a,b] \ \& \ S[a,c] \ \& \ S[a,d]$
Pattern-variables	$(xa\_)_a$	$S[a,xa\_]$
Path-variables	$\{b\}_{u\_}$	$S[u,v] \ \& \ S[v,w] \ \& \dots \& \ S[z,b]$
Null token	$(\emptyset)_a$	$S[a,\emptyset]$
Universal container	$[a]$	$S[U,a]$

Table 3.1: Annotated Parens Notation and Relational Formulas

- Containers labeled by numerals,  $(\dots)_1$ , are concrete objects, i.e. specific arrangements being considered for transformation or evaluation (Section 2.4).
- Containers labeled by small letters,  $(\dots)_a$ , are variable objects. In a rule, they can be matched to any single container in an arrangement (Section 3.2).
- Pattern-variables,  $xa\_$ , stand in place of zero, one, or many container-objects (Section 5.3.3). Names incorporate the name of the container when appropriate.
- Path-variables,  $\{\dots\}_{u\_}$ , are pattern-variables over depth of nesting, standing in place of zero, one, or many intervening nested containers (Section 9.2.1).
- The null token,  $\emptyset$ , indicates no further contents within a container (Section 4.4.1).
- The universal container,  $U$ , contains all arrangements, assuring that every arrangement has some single outermost container (Section 4.4.2).

Since the annotated parens notation is hybrid, it supports two different representations of objects, one symbolic and one iconic.

Hybrid Variable Notation:  $a \text{ =def= } (xa\_)_a$

The particular arrangement can be identified by its symbolic letter "a", or by its labeled parens form,  $(xa\_)_a$ . An arbitrary variable may contain arbitrary contents, which is indicated by the pattern variable,  $xa\_$ . When represented as a container, Object-a is  $(xa\_)_a$ . An explicitly empty container does not contain the pattern-variable.

### 3.3 Predicate Calculus

The formal language of mathematics consists of arrangements of characters in strings that follow both rules of construction and rules of interpretation. The construction of a mathematical theory begins with an intended concrete or abstract domain of discourse, the subject matter that the string descriptions refer to. We have taken arrangements of non-overlapping containers to be the domain of discourse. These arrangements are directly represented by well-formed parentheses in one-dimension and by nested circles in two-dimensions.

The symbolic theory of containment, expressed in predicate calculus, includes

- logical conjunction and quantification
- labels for container-objects
- variables that represent both collections of container-objects and arbitrary individual container-objects
- one binary relation, Contains

There are no functions and no predicates in this theory of containment.

The essential idea of predicate calculus is that particular symbolic strings can be associated with particular actual circumstances, and that manipulation of these symbolic strings can inform us about potential manipulation of actual circumstances. The domain of discourse is graced with labels for specific objects (called terms), and with labels for specifically chosen relations between objects (called atomic formulas). Predicate calculus also provides quantified variables that stand in place of objects, and logical connectives that permit assertion of logical formulas.

All formulas are subject to evaluation, they are either TRUE or FALSE. The truth assignment of a formula determines whether or not it describes actual circumstances correctly. A formal theory is a set of formulas that are all TRUE. Symbolic representation necessarily relies on the concept of truth (and

its negation, falsity) because the appearance of a symbolic token provides no specific information about its meaning. Truth is called upon in order to confirm or deny that a particular meaningless symbolic structure maps onto an actual circumstance.

The iconic notation of LoF not only incorporates a different dimension of representation, it also rests on a different, non-logical foundation. Logical truth is not central, it is not even relevant. That which is being distinguished by the logical concept TRUE is simply that which is displayed by an iconic form. An iconic form represents an actual circumstance by virtue of its appearance alone, value is directly perceivable rather than having to be interpreted.

In iconic notation, it is not necessary to illustrate that which is not TRUE. The negation of truth can be relegated to non-representation, to void. No semantic mapping function is necessary because meaning is embodied within the representation itself. Iconic form does not require labeling; the logical operators are replaced by explicit spatial relations between explicit forms. Although logical and spatial description can both be crafted to refer to the same actual circumstances, the conceptual bases upon which each description is built are fundamentally different.

### 3.4 PUT Functions

Generally, any relation can also be expressed as a function, with violations of the Existence and Uniqueness Constraints of functions handled by special techniques. The Contains relation can be expressed by the PUT function, where the putting an object into another object results in the Contains relation between the objects.

$$a \oplus b \text{ =def= PUT Object-}a \text{ into Object-}b \qquad a \oplus b \implies S[b,a]$$

Expressing containment as a function permits access to the tremendous amount of work done in function theory. Without incorporating the LoF rules, the PUT function forms a quasigroup over the set of rooted trees which represent all possible containment arrangements. The LoF rules then specify a particular variety of functional structures, one that as may be expected is not well-studied. The functional interpretation of LoF is included in Section 8 in order to connect Spencer Brown's work with conventional algebraic mathematics, and to clearly delineate it from Boolean algebras.

### 3.5 Ordered Pairs

Predicate calculus notation is intended to be descriptive, while transformation rules are dynamic. Without losing the descriptive goal, predicate calculus can be expressed in a language more appropriate for computation. This route is well explored by the Artificial Intelligence community. Formally, a relation is a set of ordered pairs. Logical conjunction in predicate calculus is replaced by set membership (a technique refined by resolution theorem proving methods), and relational formulas by ordered pairs from the domain and range of the relation (a technique refined by the list processing community). The ordered pair (a,b) reads: Object-a Contains Object-b.

Multiple Containment  $(2\ 3\ 4)_1$ :  $S[1,2] \ \& \ S[1,3] \ \& \ S[1,4]$   
 $\{(1,2), (1,3), (1,4)\}$

There is a isomorphism between the logical form (the conjunction of atomic relational formulas), and the set form (a set of ordered pairs). Conjunction maps to set membership, and formulas map to ordered pairs. However, the ordered pairs notation is logic free, which avoids potential confusion when logic itself is taken to be an interpretation of parens forms.

The descriptive set of an arrangement is the set of ordered pairs that describe that arrangement by specifying particular object-to-object containment relations. Descriptive sets can be concretely enumerated, or they can be abstracted using variables. In the ordered pairs notation, conventional variables have implicit existential quantification; a named structure must be found to exist for a rule to apply. Pattern-variables inside ordered pairs are not quantified. Pattern-variables match anything, their matching characteristic is insensitive to cardinality.

### 3.6 Directed Acyclic Graphs

A single binary relation is modeled succinctly by the mathematical object called a graph. Valid containment arrangements are graphs with finite, connected, directed and acyclic properties. Graphs are also an iconic notation, they visually display structures and relations in space. An important advantage of modeling containment by graph structures is that graphs highlight the potential for parallel processing. All nodes of a containment graph can be transformed in parallel.

A graph object is defined by two sets, a set of labels as nodes (here, names of containers), and a set of ordered pairs of labels as links (here, containment relations between container and contained). The ordered pairs description is identical to the set of graph links, with one exception. Graphs do not

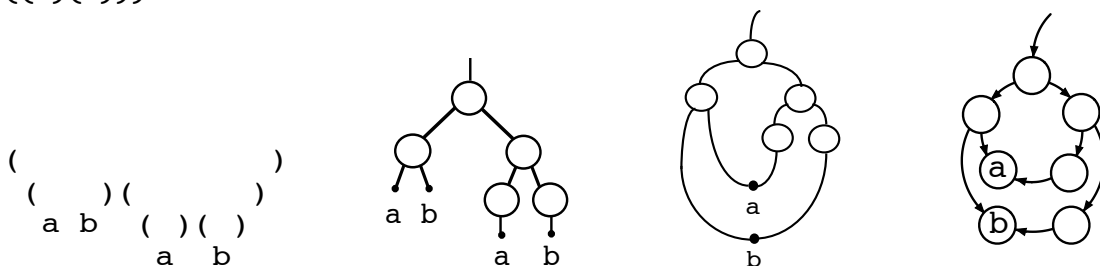


incorporate pairs that include  $\emptyset$ . An ordered pair such as  $(a, \emptyset)$  maps onto a null link; that is, such pairs do not exist within the graph representation.

A directed graph displays containment as directed links. For modeling LoF arrangements, a container is shallower, or less nested, than its contents. In graph parlance, an upper node in a directed graph is the parent node, while lower nodes are children nodes. The Contains relation is the ParentOf relation within the containment graph. Finite directed graphs also have a deepest node (sometimes called the input), equivalent to an empty container, and a shallowest node (sometimes called the output), equivalent to the universal container.

Acyclic graphs do not have loops, modeling the idea that two containers cannot mutually contain each other. An arrangement is the sub-graph of a node. That all arrangements are contained is the same as saying that the graph is connected.

The conversion of parens to graph structure is shown below for the arrangement  $((a\ b)((a)(b)))$ .



Parens are first extruded downward. Each parens is capped to construct an enclosure, and direct nesting of parens is converted into graph links. At this stage, an arrangement is represented by a rooted tree. Structure sharing combines identical labels to build a graph representation. The final circular arrangement adds directional arrows and improves the appearance.

In the graph representation, labeled nodes that stand in place of entire subgraphs must be distinguished from labeled nodes that are intended to be terminal. Variables, for example, are terminals, they do not include an internal structure. Nodes labeled by variables are input nodes of the directed graph. Named containers, in comparison, may have contents, they may identify subgraphs. We will adopt the visual convention of attaching a single downward link to possible subgraph structure. The single link stands in place of potentially zero, one, or many content links of the container node it exits from. As a short-hand for this convention, the downward link may not be labeled.

Table 3.2 illustrates the correspondence between parens, graph, relational, functional, and ordered pair notations.

Parens	$((b\ c\ d\ e)_a)_u$
Relations	$S[u,a] \ \& \ S[a,b] \ \& \ S[a,c] \ \& \ S[a,d] \ \& \ S[a,e]$
Functions	$f(f(e,f(d,f(c,f(b,a))))),u)$
Ordered pairs	$\{(u,a),(a,b),(a,c),(a,d),(a,e)\}$
Graph	

Table 3.2: Annotated Parens Notation, Relational Notation and Graph Notation

### 3.7 Variety of Notations

We will call upon several different notations, all expressing the same idea of containment. Table 3.3 shows the representation of a specific container with a single specific content object, in seven notations.

TYPE	NOTATION	REPRESENTATION
Iconic	Spencer-Brown LoF	$\overline{2}$
Iconic	Parens notation for LoF	$(2)$
Symbolic	Annotated parens	$(2)_1$
Symbolic	Relation	$S[1,2]$
Symbolic	Function	$2 \oplus 1$
Symbolic	Ordered pair	$\{ (1,2) \}$
Iconic	Directed acyclic graph	

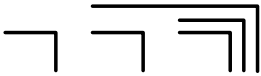
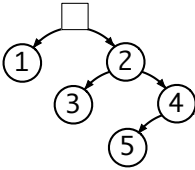
Table 3.3: Six Notations for a Specific Container

The parens version of LoF is used as a purely iconic language, and the annotated version of parens arrangements is used to bridge the gap between the iconic and symbolic notations. The predicate calculus form, and its cousin, the ordered pairs form, are both conventional symbolic notations, each with a different focus. Predicate calculus is primarily descriptive, while sets of ordered pairs can be coupled with pattern-matching to provide a computational orientation. PUT functions convert relational formulas to construction operations. Directed graphs are an alternative iconic notation that show containment in the form of connection between an upper container node and a lower contained node.

The difference between iconic and symbolic notation therefore centers on the conversion from parens to annotated parens, particularly the addition of labels and the null token, and the collapse of spatial containers into relational formulas. We'll also briefly show two other exotic varieties of iconic notation, blocks and paths, to emphasize that iconic notation is conceptually very different than the symbolic notation that is the currency of modern mathematical communication.

Table 3.4 shows an example containment arrangement in each of the seven notations.

---

Spencer Brown LoF	
Parens notation for LoF	( ) ( ( ) ( ( ) ) )
Annotated parens	$[(\emptyset)_1 ((\emptyset)_3 ((\emptyset)_5)_4)_2]$
Relational formulas	$S[U,1] \ \& \ S[U,2] \ \& \ S[1,\emptyset] \ \& \ S[2,3]$ $\& \ S[2,4] \ \& \ S[3,\emptyset] \ \& \ S[4,5] \ \& \ S[5,\emptyset]$
PUT functions	$f(1, f(f(3, f(f(5, 4), 2)), U))$
Ordered pairs	$\{(U, 1), (U, 2), (1, \emptyset), (2, 3),$ $(2, 4), (3, \emptyset), (4, 5), (5, \emptyset)\}$
Directed acyclic graph	

---

Table 3.4: An Example Arrangement in Six Different Notations

## 4 Model Constraints

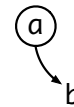
A model of containment is a specific, valid arrangement of containers. From the symbolic perspective, valid arrangements are identified by predicate calculus constraints on a universe of possible relational formulas. From the iconic perspective, valid parens arrangements are constrained by construction, they are represented directly by the set of parens structures. The relational constraints that define valid symbolic formulas simply describe the structural characteristics of iconic arrangements. That is, parens arrangements provide the semantics for symbolic expressions.

One model restriction has already been imposed, that all objects are the contents of some container. Below, the Containment Constraint is expressed as an annotated parens form, as a relational formula, and as a directed graph.

Containment:

$(b)_a$

$\exists a \exists b S[a,b]$



The Containment Constraint requires that every arrangement of containers is bounded by some outer container, eliminating side-by-side containers as valid arrangements. Arrangements, like symbolic expressions, are singular objects. Later, a universal outer container will be added to the domain of the Contains relation to assure that this constraint is met.

Since any container can contain any other container, the Contains relation is nearly universal, any pairing of container/contained is permissible except for two formal (and physically intuitive) restrictions: irreflexivity and asymmetry.

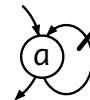
### 4.1 Irreflexive and Asymmetric

Globally across all objects in the Contains relation, an object cannot Contain itself. This Irreflexive Constraint asserts that any atomic formula of the form  $S[a,a]$  is not valid. The annotated parens form, the predicate calculus formula, and the graph notation are again shown.

Strict Containment:

$\neg(a)_a$

$\forall a \neg S[a,a]$



We will adopt the convention that a negated formula indicates an invalid formula. The negation sign in front of the parens arrangement  $(a)_a$  and in front of the formula  $S[a,a]$  indicates a containment structure that is not permitted within the conceptualization. It does not indicate that Object-a does not Contain Object-a, although this is indeed TRUE. There is no potential for

confusion between the two uses of negation because we will have no occasion to assert non-containment. Denial of containment is unnecessary since every possible arrangement includes an outer container and is thus contained. Denial of the validity of some containment relations is, however, necessary.

The heavy bar across the reentrant link in the directed graph representation indicates that the shown link is not permitted. The bar serves the same role as negation, but in a visual format. Since the self-reentrant link is to an arbitrary container, the graphical denial of linkage can be read as an existential denial,

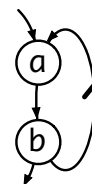
Strict Containment:  $\forall a \neg S[a,a] = \neg \exists a S[a,a]$

Although all containers can contain any other container, some specific containment relations are excluded. Locally, for specific arrangements, if a Contains b, then it is not possible that b Contains a,

No Cyclic Containment:

$(b)_a \rightarrow \neg(a)_b$

$\forall a \forall b S[a,b] \rightarrow \neg S[b,a]$



This Asymmetric Constraint forbids reentrant containment arrangements. The intention is to exclude cyclic chains of containment, a refinement to be addressed shortly. The Asymmetric Constraint eliminates all reentrant models, in other words, containment graphs are acyclic. Asymmetric relations are necessarily irreflexive.

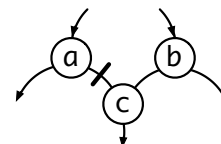
## 4.2 Physicality

In physical models, no object can be contained by two different containers, because overlap (intersection) is not physically possible when container boundaries cannot be breached. Symbolically, however, the label of an object can be replicated and placed in different containers, in violation of the no overlap restriction. Replication of labels is a primary difference between symbolic and physical models of containment.

Physical Containment:

$\neg.(c)_a \ \& \ (c)_b$

$\forall a \forall b \forall c \neg(S[a,c] \ \& \ S[b,c])$



The dot in the annotated parens form above is Quine's notation for grouping, handy when parentheses have a different use. More dots mean greater scope. The bar in the containment graph forbids a node from having more than one parent.

The Physicality Constraint is not an aspect of "perfect continence" within LoF, and is not enforced in the definitions that follow. However, it is of interest that the transformation rules of LoF precisely specify ways to eliminate (and to create) symbolic overlap.

For some strict definitions of physical containment, the deletion of a container also deletes its contents. In LoF, containers can be deleted while their contents remain. Examination of the rules in Section 7.1 shows that there is only one rule, Involution, for which this occurs.

### 4.3 Transitivity and Intransitivity

To consider transitivity of containment, two types of containment need to be distinguished. Shallow containment is not transitive; if a ShallowContains b and b ShallowContains c, then it does not follow that a ShallowContains c. In anatomy for example, a tooth socket may contain a tooth, and a tooth may contain a filling, but the tooth socket does not contain the filling. A consequence is that the ShallowContains relation,  $S[a,b]$ , is not a partial ordering.

We'll call the transitive version of containment DeepContains,  $D[a,b]$ . For example, if a shopping cart contains a box of apples, then the shopping cart DeepContains the apples, as well as ShallowContains the box they are in. In the graph representation, DeepContains is called the AncestorOf relation.

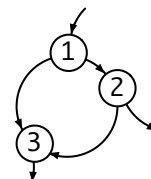
Transitive:  $\forall a \forall b \forall c \ D[a,b] \ \& \ D[b,c] \rightarrow D[a,c]$

Although the ShallowContains relation is not transitive, it is also not intransitive. For an intransitive relation, if a IsRelatedTo b and b IsRelatedTo c, then it is not the case that a IsRelatedTo c.

Intransitive:  $\forall a \forall b \forall c \ R[a,b] \ \& \ R[b,c] \rightarrow \neg R[a,c]$

The genealogical ParentOf relation is an example of an intransitive relation. However, consider the non-physical but valid arrangement  $(3 \ (3)_2)_1$ . In relational formulas and in graph notation, this arrangement would be

$(3 \ (3)_2)_1 :$   $S[1,3] \ \& \ S[1,2] \ \& \ S[2,3]$



Intransitivity would assert that given both  $S[1,2]$  and  $S[2,3]$ ,  $S[1,3]$  is not valid. The above arrangement is a counter-example. Violations of intransitivity will also be violations of the Physicality Constraint, because a contradiction of intransitivity implies that there are at least two different

containers that contain the same object. Without the Physicality Constraint, ShallowContains is neither transitive nor intransitive.

ShallowContains implies DeepContains,

Shallow Implication:  $\forall a \forall b \ S[a,b] \rightarrow D[a,b]$

The Irreflexive Constraint can now be strengthened to exclude cycles of containment,

No Cyclic Containment:  $\forall a \forall b \ D[a,b] \rightarrow \neg D[b,a]$



The vertical ellipsis in the graph represents any intervening graph structure. The iconic graph notation allows specification of arbitrary paths as well as arbitrary nodes. From the symbolic perspective, variables provide abstraction over objects, while processes such as path traversal are described iteratively. The containment graph permits both nodes and paths to be abstracted into variables.

## 4.4 Special Objects

The innermost and the outermost containers in any container arrangement are formalized next.

### 4.4.1 The Empty Container

In any finite arrangement of nested containers, the innermost will be empty, forming a ground upon which all models of containment rest. We are looking for a representation of empty containers defined solely in terms of the Contains relation.

An empty container can be represented in many different ways. Table 4.1 shows several possible varieties of representation of an empty Container-1

NOTATION	DESCRIPTION
$\bullet_1$	an atomic object
Empty[1]	a special property of an object
$S[1,\emptyset]$	a domain object with a null token in the range
$S[1,z] \ \& \ z \in \{\}$	a domain object with a range that is empty

$\neg \exists b S[1,b]$	a domain object with all range objects excluded
$S[c,1], \text{ omit } S[1,z]$	a range object not unmentioned as a domain object

---

Table 4.1: Notational Varieties for an Empty Container

None of these approaches conveys the intention of an empty container quite as well as the visual image of an enclosure without contents:  $\bigcirc$  .

The atomic container approach,  $\bullet_1$ , defines the empty container as a new type of object, one that does not participate in the domain of the Contains relation. It cannot contain because it has no interior. The empty property approach,  $\text{Empty}[1]$ , constructs a special property that attaches to each empty container. Still the empty container is not within the domain of the Contains relation. Both of these approaches portray an empty container as not a container, and both create additional descriptive mechanism that we wish to avoid.

An alternative is to construct a special token to stand in place of the absence of all objects. This token, call it " $\emptyset$ ", is what an empty container contains.

Symbolic Nothing:  $(\emptyset)_a \text{ =def?} \exists a S[a, \emptyset]$

$\emptyset$  is known as a null token, or a void reference. Although the empty container is now described by an atomic Contains formula, what it contains does not exist. The label  $\emptyset$  has no referent. Since  $\emptyset$  is not a container, this approach also introduces an exception, a token that refers to something that is not a container. That something is nothing.

From an iconic perspective, it is a bad idea to label nothing. In any image, the object of focus is brought into contrast by negative space, that is, by space that is not the object. Negative space pervades an image, there is no particular locale that supports the "nothing" label and no incentive to label the background as an object.

Another alternative is to call upon what in predicate calculus is known as the problem of vacuous truth. When  $\forall x Px$  is asserted, the assertion is true whenever the set of all  $x$  is empty.

Vacuous Nothing:  $( )_a \text{ =def?} \exists a \forall z S[a, z] \ \& \ z \in \{ \}$

Yet another alternative is to specify emptiness without labeling it by denying that the empty container has contents. That is,



Denial of Content:  $( )_a =_{\text{def}} \exists a \neg \exists z S[a, z]$

Here, the definition asserts that there does not exist an Object- $z$  that Container- $a$  Contains. No matter what  $z$  is in the range,  $a$  does not Contain it.

There is a subtle distinction in how negation is used in this approach. In predicate calculus quantification, the following transformation is valid:

Not-exists:  $\exists a \neg \exists z S[a, z] = \exists a \forall z \neg S[a, z]$

The structure  $\neg S[a, z]$  unfortunately reads:  $S[a, z]$  is not a valid Contains formula. The original intention to deny objects in the range turns into an assertion that there is an object in the domain that is excluded from participation in the Contains relation.

A final alternative is to not assert the empty object as a container. Failure to mention is not the same as either asserting or denying a relation, it is simply a convention taken to mean that a container is empty when it is explicitly a contained object yet is not explicitly a containing object. This alternative is not in the spirit of symbolic description, which presumes that all relations are explicit, but it is in the spirit of iconic description, for which relations with void are implicit.

Failure to Mention:  $\exists b \exists a S[b, a]$  and no explicit  $S[a, z]$

An empty container,  $( )$ , indirectly refers to void; the emptiness inside the empty container thus does not need labeling, it stands for itself. There's a delicate negotiation between symbolic form and the representation of nothing. What is visually obvious in the iconic form of spatial enclosure becomes conceptually turgid when expressed in symbols. Fortunately, this conceptual problem does not propagate to become a computational problem, since symbolic computation is independent of interpretation.

To reconcile symbolic and iconic approaches, we will elect to construct a special null token,  $\emptyset$ . This approach achieves the objective of making it possible to express the empty container by a valid Contains formula,  $S[a, \emptyset]$ , and is more consistent with the descriptive intent of symbolic systems. The special token can be defined as the denial of contents:

Definition of  $\emptyset$ :  $\exists a S[a, \emptyset] =_{\text{def}} \exists a \neg \exists z S[a, z]$

The null token is necessarily excluded from the domain of the Contains relation,

Not a Container:  $\neg(a)_{\emptyset} \quad \forall a \neg S[\emptyset, a]$  

In the graph representation,  $\emptyset$  is not necessary, so it does not have to be explicitly constrained. A container-node with no lower link is empty. This is represented above as a forbidden link to  $\emptyset$ .

#### 4.4.2 The Universal Container

One consequence of requiring all arrangements to be contained is that a universal outermost container,  $U$ , becomes necessary. In the annotated parens notation, this universal container is represented by square brackets,  $[ ]$ .

So that  $U$  meets the Containment Constraint, it is excluded from the range of the Contains relation,

$$\text{Not Contents:} \quad \neg(U)_a \quad \forall a \neg S[a, U] \quad \begin{array}{c} \text{X} \\ \downarrow \\ \square \end{array}$$

$U$  is not in the range, just as  $\emptyset$  is not in the domain of the Contains relation.  $U$  and  $\emptyset$  provide representational bounds for the possible arrangements of containers.  $U$  assures that all other containers are also contents.  $\emptyset$  assures that nesting can be terminated by an empty container.

By construction, the universal container DeepContains all other containers,

$$\text{Universal Container:} \quad \forall a D[U, a] \quad \begin{array}{c} \square \\ \vdots \\ \downarrow \\ \textcircled{a} \end{array}$$

In LoF Spencer Brown introduces the unwritten cross: "Suppose any  $s_0$  to be surrounded by an unwritten cross", where  $s_0$  symbolizes the shallowest space of an arrangement. The universal container renders the unwritten cross explicitly. Doing so removes much of the unconventionality in the appearance of LoF's two rules of container arithmetic.

$$\begin{array}{ll} \text{CALLING} & [ ( ) ( ) ] = [ ( ) ] \qquad [(\emptyset)_a (\emptyset)_a] = [(\emptyset)_a] \\ \text{CROSSING} & [ ( ( ) ) ] = [ \quad ] \qquad [((\emptyset)_b)_a] = [\emptyset] \end{array}$$

In the rule of Calling,  $U$  provides an outer container for the two empty containers pictured side-by-side. The side-by-side (non)arrangement,  $( ) ( )$ , can now be expressed solely in terms of containment formulas and ordered pairs,

$$\begin{array}{ll} \text{CALLING} & \forall a \quad S[U, a] \ \& \ S[U, a] \ \& \ S[a, \emptyset] \ \& \ S[a, \emptyset] \\ & = S[U, a] \qquad \qquad \& \ S[a, \emptyset] \end{array}$$

$$\begin{aligned} & \{(U,a),(U,a),(a,\emptyset),(a,\emptyset)\} \\ = & \{(U,a), \quad (a,\emptyset) \quad \} \end{aligned}$$

The rule specifies that replicated empty containers can be deleted from an arrangement. Here, the deleted formulas correspond to the arrangement  $(\emptyset)_a$ . The relational descriptions are stacked vertically to draw attention to the formulas that are deleted.

We'll call the two nested containers in Crossing, a double-container. Crossing gives permission to delete a double-container, or constructively, to add a double-container. The following relational form of Crossing will later incorporate an additional constraint that assures emptiness between the two objects forming the double-container,

$$\begin{aligned} \text{CROSSING} \quad & \forall a \forall b \quad S[U,a] \ \& \ S[a,b] \ \& \ S[b,\emptyset] \\ & = S[U,\emptyset] \\ & \{(U,a),(a,b),(b,\emptyset)\} \\ = & \{ \quad \quad \quad (U,\emptyset) \} \end{aligned}$$

The empty space on the right-hand-side of the rule of Crossing is expressed in a more conventional manner as an empty universal container,

$$\text{No Other Containers:} \quad [\emptyset] \quad \quad \quad S[U,\emptyset] \quad \quad \quad \square$$

The graph representation includes no lower link; the absence of a link expresses emptiness.

## 4.5 Arrangements

An arrangement is a valid configuration of containers. We first consider modeling the definition of an arrangement after the conventional definition of well-formed parenthesis forms [ref]:

( ) is well-formed.  
 If x is well-formed so is (x).  
 If x and y are well-formed, so is x y.

This definition in turn is modeled after the standard definition of propositions in propositional logic [ref],

Propositional variables are well-formed.  
 If P is well-formed, so is  $\neg P$ .  
 If P and Q are well-formed, so is  $P \vee Q$ .

It is the last line of the definition of parentheses that is suspect, because it constructs a well-formed structure by an operation that does not involve construction by parentheses. That is, placing  $x$  and  $y$  side-by-side introduces a conceptually separate method of construction. The definition confounds two different operations (containment and juxtaposition), and therefore does not characterize only the structure of parenthesis forms. The problem arises from trying to model parentheses after logical propositions. In the definition of propositions, logic itself provides two different operations, negation and disjunction. However, there is no reason to project these two different operators onto construction of parentheses, since parenthesis forms do not embody two different operations.

A definition of a parenthesis structure solely in terms of parentheses might state the third line as:

If  $x$  and  $y$  are well-formed, so is  $(x\ y)$ .

This description is still incorrect because it does not permit the construction of structures such as  $(x\ y\ z)$ . The definition is therefore generalized to a schema that includes any number of contained objects,

If  $x$  and  $y$  and ... are well-formed, so is  $(x\ y\ \dots)$ .

This then leads to a compact definition of an arrangement:

$(x\_)$  is an arrangement,  
where  $x\_$  stands in place of zero, one or many other arrangements.

Within LoF, Spencer Brown introduces two types of relation between distinctions (called crosses when used semantically, and tokens when used syntactically). He states in Chapter 2:

"...we have allowed only one kind of relation between crosses:  
contenance."

In the same chapter, he also states

"Call the form of a number of tokens considered with regard to one  
another (that is to say, considered in the same form) an  
arrangement."

However, "...considered with regard to one another..." is a relation between tokens/containers, and this relation is different than the contenance/Contains relation. Spencer Brown's parenthetical "...that is to say, considered in the

same form..." suggests that he understands that arrangements are always within some container.

### 4.5.1 Inductive Definition

Below, arrangements of containers are first defined inductively as parens structures. We then rely upon the annotated parens form to arrive at a relational definition.

DEFINITION	ANNOTATED PARENS
( ) is an arrangement.	$(\emptyset)_a$
If x is an arrangement, so is (x).	$(x)_a$
If x and y and ... are arrangements, so is (x y ...).	$(x\ y\ \dots)_a$
$S[a, \emptyset]$ is an arrangement.	
If x is an arrangement, so is $S[a, x]$ .	
If x and y and ... are arrangements, so is $S[a, x] \ \& \ S[a, y] \ \& \ \dots$ .	

The first line defines a ground container with no contents. The second line permits containers to be contents. The final line permits multiple contents. The incorporation of variables that represent any valid arrangement permits the definition to be interpreted recursively, thus generating all possible arrangements.

These definitions mirror the representational strategy of string structures in that they differentiate between enclosure (the second line) and juxtaposition (the third line). The iconic definition is much simpler,

$(x\_)$  is an arrangement.

The crispness of the definition, though, relies upon the use of a pattern-variable and an assumption that the contents of a container are mutually independent. This independence has already been asserted in predicate calculus as the conjunction of Contains relations, and permits a simpler relational definition,

$S[a, \emptyset]$  is an arrangement.  
If x is an arrangement, so is  $S[a, x]$ .

The grouped content-object such as  $S[a, \{x, y, z\}]$  corresponds to the juxtaposition line of the longer definition. The definition in turn indicates that  $\{x, y, z\}$  is shorthand for a conjunction of three arrangements,  $S[a, x] \ \& \ S[a, y] \ \& \ S[a, z]$ .

## 4.5.2 Construction of Arrangements

For purposes of defining possible arrangements, it is not necessary to add all of the grouped objects at the same time. That is, the set of possible arrangements can be constructed inductively from an initial container and the addition of other containment relations one formula at a time. Thus, the conjunction of relational formulas does not imply a grouping relation between objects contained by the same container. Table 4.2 shows how arrangements can be constructed one atomic formula at a time.

PARENS	ANNOTATED	ATOMIC FORMULAS
( )	( ) <sub>a</sub>	$S[a, \emptyset]$
(( ) )	(( ) <sub>a</sub> ) <sub>c</sub>	$S[c, a] \ \& \ S[a, \emptyset]$
( )	( ) <sub>b</sub>	$S[b, \emptyset]$
(( )( ))	(( ) <sub>a</sub> ( ) <sub>b</sub> ) <sub>c</sub>	$S[c, a] \ \& \ S[a, \emptyset] \ \& \ S[c, b] \ \& \ S[b, \emptyset]$
(( ( )( )) )	(( ( ) <sub>a</sub> ( ) <sub>b</sub> ) <sub>c</sub> ) <sub>d</sub>	$S[d, c] \ \& \ S[c, a] \ \& \ S[a, \emptyset] \ \& \ S[c, b] \ \& \ S[b, \emptyset]$

Table 4.2: Construction of Containment Arrangements One Formula at a Time

Conjunction of atomic formulas replaces juxtaposition of objects in the prior symbolic definition. What is implicit in this construction process is that the label of a container refers not only to that container, but also to the entire contents of the container. Above, for example, adding Container-b as contents to Container-c requires that both Container-b and its empty contents  $\emptyset$  be added to the relational description. This is nothing more than asserting that container labels refer to containers together with their contents.

Permissible sets of ordered pairs are constrained by the asymmetry of the Contains relation. This constraint identifies permissible labelings. Should the construction process always introduce new labels, the constraint is never violated. That is to say, construction by adding atomic formula with new labels enforces the intuitive and formal semantics of the Contains relation. This observation has an important implication: it is the replication of labels that is the sole notational source of potential violation of relational constraints. We have encountered this before, it is labeling that is also the source of violation of the Physicality Constraint. Labeling itself is a symptom of symbolic notation; by avoiding the necessity of labeling, iconic notation also avoids the construction of invalid forms of containment. However labeling is mandatory for any calculus of arrangements.

Arrangements can also be constructed functionally using the PUT operation. The symbolic structure  $f(a,b)$  means PUT Object- $a$  into Object- $b$ . Table 4.3 shows the functional construction of the same arrangement as in Table 4.2

PARENS	ANNOTATED	FUNCTION APPLICATIONS
( )	( ) <sub>a</sub>	
(( ) )	(( ) <sub>a</sub> ) <sub>c</sub>	$f(a,c)$
( )	( ) <sub>b</sub>	
(( )( ))	(( ) <sub>a</sub> ( ) <sub>b</sub> ) <sub>c</sub>	$f(b, f(a,c))$
(( ( ) ( ) ) )	(( ( ) <sub>a</sub> ( ) <sub>b</sub> ) <sub>c</sub> ) <sub>d</sub>	$f(f(b, f(a,c)), d)$

Table 4.3: Construction of Arrangements One Function Application at a Time

#### 4.5.2 Nested Containers

Nesting of containers is fundamental to the idea of containment, but nesting does not have a natural notation in predicate calculus (functions can be nested, but not relations). The description of nesting in predicate calculus relies upon the positioning of labels within a conjunction of relations. Nesting of relations is ill-formed,

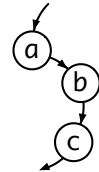
Ill-formed Formula:  $R[a, R[b, c]]$

as is the use of logical connectives within a relation,

Ill-formed Formula:  $R[a, b \ \& \ c]$

Both of these forms are not permitted because relations are defined to be mappings from object to objects, not from objects to formulas.

Nested Containment:  $((c)_b)_a$   $S[a, b] \ \& \ S[b, c]$



The logical conjunction of atomic formulas requires explicit replication of labels in order to flatten the nested structure that is visually apparent in both the parens and the graph forms, neither of which require replicated labels. Symbolic labels indicate a pattern, while the parens and graph forms show that pattern explicitly.

Table 4.4 uses a hybrid annotated parens notation to show that different conjunctions of relations correspond uniquely to different arrangements of containers.

CONJUNCTION	ANNOTATED PARENS	INTERPRETATION
$(1)_1$	--	disallowed, no self-containment
$(1)_2 \& (2)_1$	--	disallowed, no cyclic containment
$(1)_2 \& (1)_2$	$(1)_2$	replication/idempotency of conjunction
$(1)_2 \& (1)_3$	$(1)_2$	replication/structure sharing
$(1)_2 \& (2)_3$	$((1)_2)_3$	nesting
$(1)_3 \& (2)_3$	$(1\ 2)_3$	multiple contents, same container
$(1)_2 \& (3)_4$	$(1)_2 (3)_4$	multiple contents, outer container missing

Table 4.4: Arrangements and Conjunction of Relational Formulas

Within the symbolic representation, every separate object in the contents is specified by a separate atomic formula. Multiple contents are expressed by the conjunction of formulas. Nesting is expressed by the same label occurring in both the domain and the range of formulas joined by conjunction. The null token identifies empty containers.

## 4.6 Containment Revisited

Two special objects,  $U$  as an always present outer container, and  $\emptyset$  as a null token, have been added. And the concept of content-objects has been eliminated, in favor of a conjunction of simple relational formulas. These changes can be expressed within the formal definition of the ShallowContains relation:

Let  $C$  be a finite set of containers, labeled  $\{1,2,3,\dots\}$ ,  
and  $U$  be a special container,  
and  $\emptyset$  be a special token.

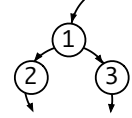
The ShallowContains relation  $S$  is on  $C \cup \{U\}$  to  $C \cup \{\emptyset\}$ , and is a subset of  $C \cup \{U\} \times C \cup \{\emptyset\}$ .



Technically, containment must be a relation rather than a function. Both the Existence and the Uniqueness Constraints of functions are not satisfied. The token without a referent,  $\emptyset$ , was constructed to satisfy the Existence Constraint, since an empty container in the domain fails to map onto any container-object in the range. The conjunction of object-to-object relations explicitly denies the Uniqueness Constraint, since a container of many objects maps one-to-many onto the range.

Multiple Containment:  $(2\ 3)_1$

$S[1,2] \ \& \ S[1,3]$



The original definition of containment that included content-objects meets the Uniqueness Constraint, and can now be seen to reflect functional modeling. Since ShallowContains implies DeepContains, DeepContains is also not a function.

Table 4.5 shows the relational constraints that define the intended interpretation of containment, expressed in both the annotated parens and relational notations.

CONSTRAINT	ANNOTATED PARENS FORM	RELATIONAL FORM
Containment	$(b)_a$	$\forall b \exists a \ S[a,b]$
Irreflexive	$\neg(a)_a$	$\forall a \ \neg S[a,a]$
Asymmetric	$(b)_a \rightarrow \neg(a)_b$	$\forall a \forall b \ S[a,b] \rightarrow \neg S[b,a]$
Empty container	$(\emptyset)_a$	$\exists a \ S[a,\emptyset]$
Not a container	$\neg(a)_\emptyset$	$\forall a \ \neg S[\emptyset,a]$
Not contents	$\neg(U)_a$	$\forall a \ \neg S[a,U]$
Physicality	$\neg.(c)_a \ (c)_b$	$\forall a \forall b \forall c \ \neg(S[a,c] \ \& \ S[b,c])$

Table 4.5: Symbolic Constraints that Define the Meaning of Containment

These constraints are placed on labeled symbolic forms, they are intended to describe the actual arrangements that are embodied in the unlabeled iconic forms.

## 5 Iconic Algebra

An algebra permits transformation of structure by asserting equality between expressions in a well-defined language. Equations specify how an expression can change without changing the meaning or intention or value of that expression. Variables permit equations to address classes of expressions. Change is

implemented by substitution. Meaning is also defined by equations, by those equations that are taken as the assumptions (called initials and axioms) upon which the algebra is built.

LoF is an iconic algebraic system. Interpreted operationally, the equations of LoF assert valid transformations of arrangements of spatial containment. By equating patterns of containment, compound arrangements can be reduced to simple arrangements. Crossing and Calling reduce all compound arrangements to one of two simple arrangements,  $[( )]$  or  $[ \quad ]$ .

## 5.1 The Computational Algebra

The LoF arithmetic consists of Crossing and Calling, both of which are generalized by the LoF algebra. The LoF calculus rests on a basis of three computational rules,

DOMINION	$a ( ) = ( )$
INVOLUTION	$((a)) = a$
PERVASION	$a (a b) = a (b)$

Each rule identifies structure in arrangements of parens that can be deleted without changing the intention of that arrangement. Each rule shows structure on the left-hand-side that is deleted on the right-hand-side. Such structure is void-equivalent, within the entire arrangement its presence is irrelevant. The identification of void-equivalent structure by pattern-matching characterizes LoF computation. The Principle of Void-Equivalence assures that LoF computation is sound,

Structures that are equivalent to void are semantically inert and syntactically irrelevant.

Dominion is the termination condition, the last step of a reduction. As a process, Dominion eliminates irrelevant context. Involution rids a representation of superfluous double-containers. Pervasion contributes almost all of the computational effort, yet still simply deletes unnecessary replication from pervaded spaces.

These rules are both sufficient and efficient. Each rule identifies unlabeled patterns of containment; variables are needed only to identify possible incidental structure within a larger arrangement that includes the pattern. LoF variables generalize the applicability of the rules as transformations. Dominion and Involution incorporate only one variable. Pervasion uses Variable-b in a passive role, to represent what remains after replicates of Variable-a have been deleted.

## 5.2 Equations

We are addressing three notions of an equation,

- within an algebra of well-formed expressions that asserts structures belonging to the same equivalence classes,
- within a logic of relations that maintains logical value under substitution, and
- within an iconic calculus of pattern-matching that permits transformation of structure via void-substitution.

An algebraic equation asserts that expressions joined in equality have equivalent values. Algebraic equations also include variables that stand in place of arbitrary expressions. Expressions are singular objects.

The intention of a predicate calculus equation is to identify substitutable expressions, regardless of their value. Substitution is assured to maintain the value of the changed expression. Predicate calculus equations then identify structural changes that do not impact the value of the entire expression.

Equations in LoF are stated in terms of existent structural patterns, together with variables that stand in place of zero, one or many arbitrary arrangements. These equations identify patterns within an arrangement that are irrelevant to the intention of that arrangement. They give permission to delete the irrelevant structure. Crossing identifies double-containers as irrelevant; Calling identifies replicated containers as irrelevant.

Logical deduction relies upon implication as well as equality. Since LoF is an algebra, it does not incorporate implication as a primary connective. However, LoF is not a standard algebra since it incorporates pattern-variables. The iconic nature of LoF arrangements includes both spatial representation and parallel transformation. For that reason, LoF might be described as a parallel graph rewrite system. Several example of graph rewriting are illustrated in the Appendix.

From the perspective of predicate calculus, the LoF inference rules are limited to the rules that govern the theory of equivalence relations. This theory defines equivalence by three properties of the binary equality relation, Reflexivity, Symmetry and Transitivity; and two methods of transformation, Replacement and Substitution. Table 5.1 shows the formal definition of the theory of equivalence. Spencer Brown establishes these definitions and rules within the LoF text.

---

Properties of equality:	
Identity	$\forall x \ x=x$
Symmetry	$\forall x \forall y \ x=y \rightarrow y=x$
Transitivity	$\forall x \forall y \forall z \ x=y \ \& \ y=z \rightarrow x=z$
Transformation rules:	
LoF Replacement	$\forall x \forall y \forall a \forall b \ x=y \rightarrow (\text{subst } a \ b \ x) = (\text{subst } a \ b \ y)$
LoF Substitution	$\forall x \forall y \forall a \ x=y \rightarrow a=(\text{subst } x \ y \ a)$

---

Table 5.1: The Theory of Equality

The abbreviation (subst x y z) stands for Substitute x for y in z.

### 5.3 Transformation Rules

Annotated parens forms include labels that bridge to a symbolic description, however additional annotation is needed to unambiguously identify patterns in symbolic notation that are eligible for transformation.

LoF rules are unique in that they identify specific component patterns that are irrelevant to intended meaning, and thus can be deleted. Deletion is achieved by void-substitution, substituting literally nothing for the identified structure. Most conventional transformation systems specify rearrangements rather than deletions. The application of LoF rules occurs via void-substitution, void is substituted for an irrelevant pattern. Constructively, irrelevant patterns can be substituted for void any place within an arrangement. In symbolic notations, void occurs only in between entire symbols (words).

Involution serves as an example of a LoF transformation rule. The Involution rule gives permission to delete a double-container.

INVOLUTION                       $((a)) = a$

Involution applies in all cases of a double-container. Double-containers can be deleted or constructed at will. Variable-a is a pattern-variable that stands in place of zero, one or many objects. In the case that Variable-a stands in place of zero arrangements, Involution becomes Crossing,

CROSSING                       $(( \ )) =$

An application of Crossing deletes or constructs an empty double-container anywhere within an arrangement. For example,

( ( ( ) ) )	initial arrangement
( ( ) )	pattern matches the Crossing rule, ( ( ) ) =
( )	substitute void for the matched pattern

Above, a component pattern within the initial arrangement matches the Crossing rule. The match means that the double-container can be deleted. Alternatively, Crossing can be applied constructively.

( )	initial arrangement
( ( ) )	pattern matches the Crossing rule, = ( ( ) )
( ( ( ) ) )	substitute ( ( ) ) for the matched pattern

The iconic void has no equivalent in symbolic notations. In iconic notations, space, or void, permeates the entire form. It exists everywhere; it cannot be replicated; and it is not "replaced" during a substitution.

## 5.4 The Description of Transformation

The relational description of containment arrangements consists of a conjunction of atomic formulas. As an example, the initial arrangement above is described as

( ( ( ) ) ) :                      ((( $\emptyset$ )<sub>3</sub>)<sub>2</sub>)<sub>1</sub>                      S[1,2] & S[2,3] & S[3, $\emptyset$ ]

For the purposes of description, the relational formulas are sufficient. However, the symbolic description of a pattern-based transformation rule needs to identify four different types of structure,

- explicitly necessary structure
- structure that is contextually necessary but not changed
- structure that is incidental to the transformation, and
- structure that when present blocks a transformation

Explicit structure is explicitly present in the rule. Contextual structure is present in the rule but passed through a transformation unchanged. Incidental structure is not present in a rule and is passed through a transformation unchanged. Forbidden structure is present in a rule and causes a rule application to fail when found to be present in the arrangement being transformed.

### 5.4.1 Exactly One

Crossing asserts that double-containers can be deleted.

$$\text{CROSSING} \quad ((\ )) = \quad [(\emptyset (\emptyset)_c)_b] = [\emptyset]$$

In annotating this transformation rule, a null token specifies that the innermost of the container pair is empty. In order to identify a double-container explicitly, the outermost of the container pair must have exactly one object as its contents.

$$\text{Double-container:} \quad ("no\ others" (\emptyset)_c)_b$$

The assertion of "no others" in the presence of one object is known as uniqueness quantification in predicate calculus.

$$\text{Exactly One:} \quad \exists b \exists c \forall x \ S[b, c] \ \& \ S[b, x] \rightarrow c=x$$

This reads: given that b Contains c, if b Contains anything else, that something else is identical to c.

We wish to specify quantification of exactly one without the incidental identity relation between labels in the above definition. This can be accomplished by extending the role of the null token to stand in place of any existential limitation of contents.

$$\text{Double-container:} \quad (\emptyset (\emptyset)_c)_b \quad S[b, \emptyset] \ \& \ S[b, c] \ \& \ S[c, \emptyset]$$



This reads: b Contains c, and nothing else; c is empty. The relational form uses the null token in both an absolute and a relative sense. The graph and parens representations require neither.

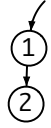
### 5.4.2 Nothing Else

The specification of exactly one typifies a generic issue: the description of spatial transformation in symbolic languages requires specification of what is not there as well as what is there. What is not there is visually, but not textually, obvious. The root of this issue lies in the promiscuous replication of labels permitted in symbolic notations, which results in a necessity to label everything, including nothing!

The description of a double-container does not require the relative null token, since convention establishes that if it is not explicit, it is not an aspect of the actual circumstances being described. The absolute null token,

however, is required so that the emptiness of the inner container can be expressed as an atomic formula.

A Double-container:  $((\emptyset)_2)_1$   $S[1,2] \ \& \ S[2,\emptyset]$



A difficulty arises, however, for transformation rules. Description can default to explicit structure, but rule application is asking a question about the presence of an explicit structure, not making a descriptive declaration. The double-container in the rule of Crossing requires the relative null token to specify that the outer container have exactly one object as contents,  $(\emptyset (\emptyset))$ , for the rule to apply.

The necessity of the relative null token can be illustrated using two arrangements, one for which Crossing does apply, another for which it does not. We show the Crossing rule applied to the iconic transformation of annotated parens arrangements,

Arrangement-A

$((\emptyset)_3)_2)_1$   
 $(( \ ) \ )$  match Crossing  
 $( \ \emptyset \ \ )_1$  apply Crossing

Arrangement-B

$((\emptyset)_3 (\emptyset)_4)_2)_1$   
 $(( \ ) \ \ )$  no match  
 $((\emptyset)_3 (\emptyset)_4)_2)_1$  no change

The ordered pair representation of these arrangements is

Arrangement-A:  $\{(1,2) (2,3), (3,\emptyset)\}$

Arrangement-B:  $\{(1,2), (2,3), (3,\emptyset), (2,4), (4,\emptyset)\}$

Consider now two versions of the Crossing rule that differ only in the presence of the relative null token.

Crossing-I:  $(( (\emptyset)_c)_b)_u = (\emptyset)_u$   $\{(u,b), (b,c), (c,\emptyset)\} = \{(u,\emptyset)\}$   
 Crossing-II:  $((\emptyset (\emptyset)_c)_b)_u = (\emptyset)_u$   $\{(u,b), (b,c), (b,\emptyset), (c,\emptyset)\} = \{(u,\emptyset)\}$

Match each arrangement with the Crossing-I rule, in a fashion similar to the iconic pattern match above,

Arrangement-A:  $\{(1,2), (2,3), (3,\emptyset)\}$   
 Crossing-I:  $(u,b) (b,c) (c,\emptyset)$  succeed  $u/1, b/2, c/3$   
 Arrangement-B:  $\{(1,2), (2,3), (3,\emptyset), (2,4), (4,\emptyset)\}$   
 Crossing-I:  $(u,b) (b,c) (c,\emptyset)$  succeed  $u/1, b/2, c/3$

Crossing-I incorrectly matches both arrangements. In both cases, Variable-u matches 1, Variable-a matches 2, and Variable-b matches 3. The unmatched pairs in Arrangement-B are not excluded explicitly, they are seen by the rule application as incidental structure that is irrelevant to the rule. But this particular structure should cause the rule to fail, while incidental structure found elsewhere should not. Crossing II provides the appropriate filter.

Arrangement-A:	$\{(1,2),(2,3),(3,\emptyset)\}$	
Crossing-II:	$(u,b) (b,c) (c,\emptyset) (b,\emptyset)$	succeed $u/1,b/2,c/3$
Arrangement-B:	$\{(1,2),(2,3),(3,\emptyset),(2,4),(4,\emptyset)\}$	
Crossing-II:	$(u,b) (b,c) (c,\emptyset) (b,\emptyset)$	fail

Recall that the null token is defined as an assertion that there are no objects in the range that match it. In Crossing-II, Variable-u still matches 1, Variable-b matches 2 and Variable-c matches 3. In Arrangement-A, the filter pair  $(b,\emptyset)$  fails to match, that is, there are no further contents of Container-b. The rule succeeds. However in Arrangement-B, the  $(b,\emptyset)$  pair aligns with the  $(2,4)$  pair. This is a structural mismatch since Object-4 is in the range while  $\emptyset$  declares that there can be no such objects. As a result the rule application fails.

Several examples of rule application by pattern-matching are provided in the Appendix. The above discussion provides an initial justification for an important contrast. The relational description of transformation rules must include both positive and negative conditions for matching. Positive conditions specify the structures that must be present, negative conditions specify structures that must not be present. In contrast, the iconic description of both rules and arrangements uses empty space (i.e. non-representation) to specify negative conditions.

### 5.4.3 Context-Free Rules

LoF transformations are intended to apply at all depths of nesting. Consider Crossing, for example, which applies whenever double-containers occur within an arrangement. As currently formulated, the rule is limited to the shallowest space of an arrangement, the contents of U. The rule, as it stands, is also not specific about any other contents within the universal container.

CROSSING	$(( \ )) =$	$[(\emptyset (\emptyset)_c)_b] = [\emptyset]$
	$\{(U,b),(b,\emptyset),(b,c),(c,\emptyset)\}$	
	$= \{ \quad \quad \quad (U,\emptyset) \}$	



A minor modification to the current notation can accommodate the generalization to depth-free rules. Let the universal container be any arbitrary container. To do so, replace U with a variable, u, that stands in place of any single container. Here, for example, Crossing is generalized to any depth,

$$\begin{aligned} \text{CROSSING} \quad ((\ )) = \quad & ((\emptyset (\emptyset)_c)_b)_u = (\emptyset)_u \\ & \{(u,b), (b,\emptyset), (b,c), (c,\emptyset)\} \\ & = \{ \quad \quad \quad (u,\emptyset) \} \end{aligned}$$

The Crossing rule also applies regardless of other contents within the outer Container-u. The presence of Object-1 in the arrangement (1 (( ))), for example, should not to interfere with the rule application. Again, there is a variety of choices about how to handle this. An explicit technique is to include a pattern-variable within the contents of the outer container.

Pattern-variables stand in place of arbitrary patterns. Pattern-variables are represented by a labeled underbar, x\_, to emphasize that the variable can be associated with any and all contents of a given container. We'll use small letters from the end of the alphabet as pattern-variables. When desirable, pattern-variables incorporate the name of their container, as in (xu\_)\_u.

The Crossing rule becomes,

$$\begin{aligned} \text{CROSSING} \quad ((\ )) = \quad & (xu_-(\emptyset (\emptyset)_c)_b)_u = (xu_-)_u \\ & \{(u,xu_-), (u,b), (b,\emptyset), (b,c), (c,\emptyset)\} \\ & = \{(u,xu_-) \quad \quad \quad \} \end{aligned}$$

Containers that are identified explicitly (named in the annotation) within a LoF transformation rule represent the pattern itself. Pattern-variables bind to components of a structure that do not match the pattern. This type of variable then, identifies the rest of the content in a container. For example, the Involution rule specifies a double-container pattern, with any collection of objects, including none, inside (bound to y\_) or outside (bound to xu\_) the double-container.

$$\begin{aligned} \text{INVOLUTION} \quad ((a)) = a \quad & (xu_-(\emptyset (y_-)_c)_b)_u = (xu_- y_-)_u \\ & \{(u,xu_-), (u,b), (b,\emptyset), (b,c), (c,y_-)\} \\ & = \{(u,xu_-), \quad \quad \quad (u,y_-)\} \end{aligned}$$

#### 5.4.4 Incidental Structure

From the perspective of transformation by pattern-matching, the above pattern-variable  $x_$  is not necessary. Incidental structure can be managed instead by a convention to ignore arrangements that do not directly participate in a rule. When an ordered pair does not directly participate in a pattern-based transformation rule, then that pair simply remains as part of the structural description. Non-participating pairs do not need to be represented by a variable and do not need to be accounted for in a rule. This particular convention is adopted within both LoF and predicate calculus. With this convention, the prior relational specification of Involution can be modified by eliminating the  $xu_$  pattern-variable:

$$\text{INVOLUTION} \quad ((a)) = a \quad (xu_-(\emptyset (y_-)_c)_b)_u = (xu_- y_-)_u$$

$$= \{ \begin{matrix} \{(u,b), (b,\emptyset), (b,c), (c,y_-)\} \\ (u,y_-) \end{matrix} \}$$

The pattern-variable  $y_$  cannot be eliminated from the symbolic description because the contents of Container- $c$  do participate in the symbolic transformation. They change their container from  $c$  to  $u$ . We retain pattern-variables within the annotated parens forms as explicit reminders of the potential presence of incidental arrangements.

## 6 The Role of Variables

A bridge between relational and iconic descriptions has been constructed by annotating the parens notation to include pattern-variables that do not distinguish absence of structure, uniqueness of structure, or multiplicity of structure. Pattern-variables occur only in transformation rules. This type of variable translates into predicate calculus as a rule schema. When a container has two pattern-variables as contents, the pair represents all possible partitions of the content.

### 6.1 LoF Pattern-Variables

The variables in LoF rules are pattern-variables that match an arbitrary collection of container-objects. For example, within the rule of Involution,

$$\text{INVOLUTION} \quad ((a)) = a \quad (xu_-(\emptyset (y_-)_c)_b)_u = (xu_- y_-)_u$$

the variable  $y_$  can be matched by the absence of an arrangement, by a single arrangement, and by more than one arrangement. Here is an example of a valid application of Involution, in iconic notation:

((1 2 3))	initial arrangement
(( a ))	rule match , a = 1,2,3
1 2 3	rule application

In the next example, the pattern-variable matches empty space,

(( ))	initial arrangement
(( a ))	rule match , a =
	rule application

It is unconventional in mathematical notation to permit the absence of an object to be bound to a variable, and it is also unconventional to permit more than one object to be bound to a particular variable. Pattern-variables are common within the computational community today, but were quite novel when LoF was published. In 1967, pattern-variables were used only in the computer programming research language SNOBOL.

Spencer Brown defines variables as standing in place of valid LoF expressions, and absence is identified as a valid expression. We have varied from LoF by requiring absence to be framed by an explicit container. This variation is necessary only in providing tools to transcribe LoF into symbolic notations. LoF rules reduce arrangements to one of two equivalence classes, one being void. A variable can therefore stand in place of void within the course of a case analysis of the possible values of an arrangement.

Similarly, multiple arrangements are defined as valid LoF expressions in the text, whereas we require that they be contained by some container-object. This is necessary again only in constructing a symbolic representation of containment.

Table 6.1 presents this final interpretation of the use of constants and variables in the annotated parens forms. Path variables are described in Section 9.2.

REPRESENTATION	TYPE	USE	MEANING
$\emptyset$	Null token	imaginary constant	nothing else
[...]	Universal container	imaginary container	contains all objects
(...) <sub>a</sub>	Labeled container	conventional variable	arbitrary object
x <sub>-</sub>	Pattern-variable	any pattern	zero, one or many
{...} <sub>x<sub>-</sub></sub>	Path-variable	any depth	any path

Table 6.1: Constants and Variables in Annotated Parens Notation

## 6.2 Variables That Partition

We now consider the intended meaning of LoF rules that have two pattern-variables within one container. The archetype is the Pervasion rule:

$$\text{PERVASION} \quad a (a \ b) = a (b) \quad (x\_ y\_ (y\_ z\_ )_c)_u = (x\_ y\_ (z\_ )_c)_u$$

The arrangement  $(a \ b)$  in Pervasion has broad implications since the two LoF variables stand in place of all possible partitions of the contents of the container. For example, each of these rule applications is a single application of the Pervasion rule,

RULE APPLICATION	PATTERN-MATCH
$1 (1 \ 2) = 1 (2)$	$a = 1 \quad b = 2$
$1 (1 \ 2 \ 3 \ 4) = 1 (2 \ 3 \ 4)$	$a = 1 \quad b = 2 \ 3 \ 4$
$1 \ 2 (1 \ 2 \ 3 \ 4) = 1 \ 2 (3 \ 4)$	$a = 1 \ 2 \quad b = 3 \ 4$
$1 \ 2 \ 3 (1 \ 2 \ 3) = 1 \ 2 \ 3 ( )$	$a = 1 \ 2 \ 3 \quad b =$

In Pervasion, the replicated Variable- $a$  stands in place of any collection of the contents of the container, while the non-replicated Variable- $b$  stands in place of what remains when the collection labeled  $a$  is removed. The only objects that are relevant to the rule are those with replicated labels. If replicated labels are considered to be pointers to unique objects, then partitioning is implemented as simple matching rather than as search.

## 6.3 Rule Schemas and Set-Variables

A rule schema, or pattern template, is an abstract rule that succinctly represents a collection of similar more specific rules. Rule schemas are common in predicate calculus, in fact, inference rules themselves are schemas that identify an infinite number of specific cases. The Involution rule would be considered to be a rule schema that stands in place of many specific rules, one for each different specific number of objects in the content of the double-container:

INVOLUTION-0	$(( \ )) =$	$a =$
INVOLUTION-1	$((1)) = 1$	$a = 1$
INVOLUTION-2	$((1 \ 2)) = 1 \ 2$	$a = 1, 2$
INVOLUTION-3	$((1 \ 2 \ 3)) = 1 \ 2 \ 3$	$a = 1, 2, 3$
...		

From the perspective of LoF pattern-variables, it does not matter how many objects are contained by the double-container because the rule application is intended to be independent of those contents. In symbolic notation, however, the contents of the double-container are not independent, since these arrangements change their labeled containers. The adjustment is one of relabeling and is thus necessary only within the symbolic representation.

Table 6.2 shows specific examples of the Involution rule schema in annotated parens notation and in relational notation.

CONTENTS	ANNOTATED PARENS	ORDERED PAIRS
None	$((\emptyset (\emptyset)_2)_1)_u = (\emptyset)_u$	$\{(u,1),(1,\emptyset),(1,2),(2,\emptyset)\}$ = { (u, $\emptyset$ ) }
One	$((\emptyset (3)_2)_1)_u = (3)_u$	$\{(u,1),(1,\emptyset),(1,2),(2,3)\}$ = { (u, 3) }
Two	$((\emptyset (3\ 4)_2)_1)_u = (3\ 4)_u$	$\{(u,1),(1,\emptyset),(1,2),(2,3),(2,4)\}$ = { (u, 3), (u, 4) }
Multiple	$((\emptyset (3\ 4\ \dots)_2)_1)_u = (3\ 4\ \dots)_u$	$\{(u,1),(1,\emptyset),(1,2),(2,3),(2,4),\dots\}$ = { (u, 3), (u, 4), ... }

Table 6.2: The Involution Rule Schema

Pattern-variables are not native to predicate calculus. They can, however, be seen as a convenient abbreviation for more explicit predicate calculus forms. To convert pattern-variables to their predicate calculus counterparts, begin by constructing different set-variables for each pattern-variable. The set-variable replaces those objects collected by the pattern-variable with a set containing those objects. Then unpack the set-variable into a conjunction of relations.

$$S[u, x_] \implies S[u, \{1, 2, \dots\}] \implies S[u, 1] \ \& \ S[u, 2] \ \& \ \dots$$

Since different sets can have different cardinalities, the unpacking of set variables is a schema that stands in place of many specific cases, each with an explicit cardinality.

## 6.4 Principle of Relevance

Spencer Brown's Principle of Relevance is

"If a property is common to every indication, it need not be indicated."

He calls this principle into play, for example, in not recording the unwritten cross, what we have made explicit as the universal container. Similarly, he does not record the null token potentially present in every container to specify “nothing else”.

The annotated parens notation can now be converted by steps into the LoF notation. This deconstruction shows precisely where the Principle of Relevance has been applied, and provides a map of how symbolic notation has been added to the iconic form. In Table 6.3, Pervasion serves as an example.

PERVASION	$a \ (a \ b) = a \ (b)$	$(x\_ \ y\_ (y\_ \ z\_))_c)_u = (x\_ \ y\_ (z\_))_c)_u$
	$(x\_ \ y\_ (y\_ \ z\_))_c)_u = (x\_ \ y\_ (z\_))_c)_u$	annotated parens
	$x\_ \quad \quad \quad x\_$	remove incidental structure
	$( \quad \quad \quad )_u \quad ( \quad \quad \quad )_u$	remove explicit depth-free
	$\quad \quad \quad c \quad \quad \quad c$	remove container label
	$\quad \quad \quad - \quad - \quad - \quad \quad \quad - \quad -$	remove pattern-variable notation
	$y \ (y \ z) = y \ (z)$	remaining
	$a \ (a \ b) = a \ (b)$	relabel

Table 6.3: Deconstructing Annotated Parens Notation

The incidental structure pattern-variable  $x_{\cdot}$  and the depth-free notation  $(\dots)_u$  are both characteristic of all LoF rules, so Spencer Brown does not record them. Since the containers are referenced by their participation in a particular arrangement, they do not need to be labeled. We added notation for pattern-variables in order to compare iconic and symbolic notations. Since all variables in LoF are pattern-variables, the Principle of Relevance permits deletion of the specialized notation. What remains is an elegant notation that expresses the iconic structure of LoF.

## 7 Computation

The LoF rules are described in both iconic and symbolic notations. For all notations, a rule applies to a particular arrangement only when the component structures match. Incidental structure that does not match, remains and is unchanged. Detailed examples of computation via pattern-matching are presented in the Appendix.

We will discuss some additional problems of converting the iconic representation of containment to a symbolic language after presenting the LoF rules. The modeling issues that will remain include inherent parallelism, deep rules (i.e. rules expressed using DeepContains rather than ShallowContains), structure sharing, void-based computation, and notational variants of iconic languages (Section 9).

### 7.1 Iconic and Symbolic Calculus

The LoF transformation rules are presented in five notations: parens, annotated parens, sets of ordered pairs, relational logic, and directed acyclic graphs.

Table 7.1 shows the rules of LoF in iconic parens notation. Spencer Brown identifies two initials for the arithmetic and two initials for the algebra. Table 7.1 also shows an alternative set of three initials that are particularly convenient for computation. These three computational initials are void-based, they implement transformation solely by construction and deletion of irrelevant structure. Spencer Brown's Transposition, in contrast, is based on rearrangement.

---

CROSSING CALLING	$(( )) =$ $( ) ( ) = ( )$	initials of the arithmetic
POSITION TRANSPPOSITION	$(a (a)) =$ $((a b)(a c)) = a ((b)(c))$	initials of the algebra
DOMINION INVOLUTION PERVASION	$a ( ) = ( )$ $((a)) = a$ $a (a b) = a (b)$	computational initials
REPLICATION OCCLUSION	$a a = a$ $(a ( )) =$	theorem theorem

---

Table 7.1: The Iconic Transformation Rules of LoF

Table 7.1 shows two theorems. Replication is a generalization of Calling, while Occlusion is a generalization of Crossing.

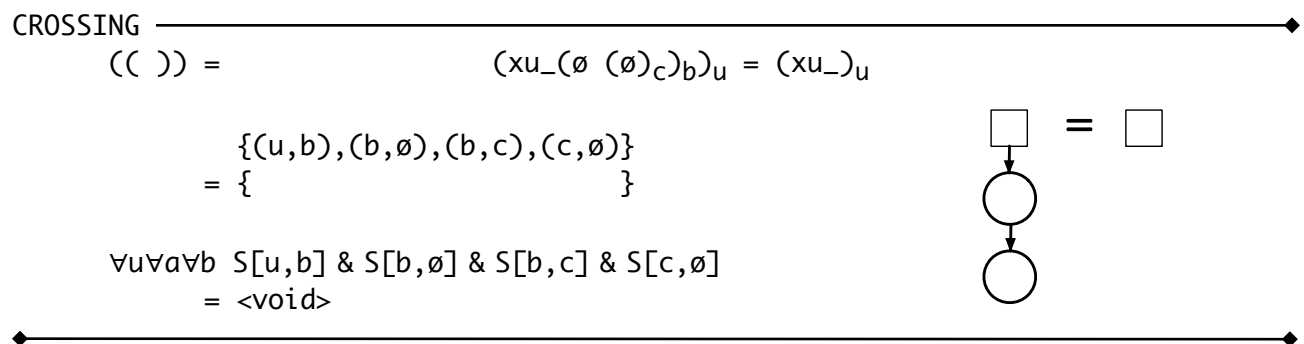
What is innovative in the LoF notation for transformation rules can be summarized as,

- The language consists of images rather than symbols.
- The iconic representation combines meaning and representation.
- Computation proceeds by deletion of irrelevant structures.
- The pattern-based rules apply in any context.
- Variables bind to zero, one or more objects.
- Objects that define patterns represent themselves, and do not require labeling.
- Objects are independent rather than connected. The iconic notation is naturally parallel.

For the rules that follow, the relational and the ordered pair descriptions are both stacked vertically to emphasize the formulas (respectively, pairs) in each transformation that do not change. These catalytic formulas define the contexts that permit each rule to be applied.

### 7.1.1 Initials of the Arithmetic

The first two transformation rules are the initials of the arithmetic of form presented in LoF.



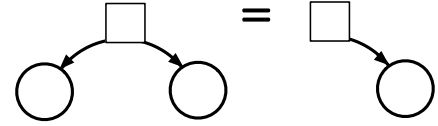


CALLING ◆

$$( ) ( ) = ( ) \quad (xu_{-}(\emptyset)_a(\emptyset)_a)_u = (xu_{-}(\emptyset)_a)_u$$

$$\begin{aligned} & \{(u,a), (u,a), (a,\emptyset), (a,\emptyset)\} \\ &= \{(u,a), \quad (a,\emptyset) \quad \} \end{aligned}$$

$$\begin{aligned} \forall u \forall a \quad & S[u,a] \ \& \ S[u,a] \ \& \ S[a,\emptyset] \ \& \ S[a,\emptyset] \\ &= S[u,a] \quad \quad \& \ S[a,\emptyset] \end{aligned}$$



Pattern-variables combine Crossing and Involution into a single rule. Similarly, Calling and Replication can be combined. Another way to generalize Crossing is the Occlusion theorem, which is also closely related to both Position and Dominion, described in the next section.

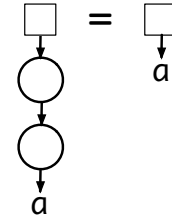
Thus, Crossing is both a special case of Involution and a special case of Occlusion. Calling is a special case of Replication. Replication is a LoF theorem that Spencer Brown calls Iteration. Occlusion is not listed within the text as a theorem.

INVOLUTION ◆

$$((a)) = a \quad (xu_{-}(\emptyset (y_{-})_c)_b)_u = (xu_{-} y_{-})_u$$

$$\begin{aligned} & \{(u,b), (b,\emptyset), (b,c), (c,y_{-})\} \\ &= \{ \quad \quad \quad (u,y_{-}) \quad \} \end{aligned}$$

$$\begin{aligned} \forall u \forall a \forall b \quad & S[u,b] \ \& \ S[b,\emptyset] \ \& \ S[b,c] \ \& \ S[c,y_{-}] \\ &= \quad \quad \quad S[u,y_{-}] \end{aligned}$$

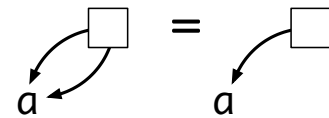


REPLICATION ◆

$$a \ a = a \quad (x_{-} y_{-} y_{-})_u = (x_{-} y_{-})_u$$

$$\begin{aligned} & \{(u,y_{-}), (u,y_{-})\} \\ &= \{(u,y_{-}) \quad \quad \} \end{aligned}$$

$$\begin{aligned} \forall u \quad & S[u,y_{-}] \ \& \ S[u,y_{-}] \\ &= S[u,y_{-}] \end{aligned}$$



OCCCLUSION ◆

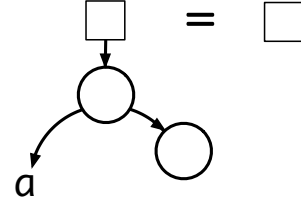
$$(a ( )) = \quad \quad \quad (x_{-}(y_{-}(\emptyset)_c)_b)_u = (\emptyset)_u$$

$$\{(u,b),(b,y_-),(b,c),(c,\emptyset)\}$$

$$= \{ \quad \quad \quad \}$$

$$\forall u \forall b \forall c \ S[u,b] \ \& \ S[b,y_-] \ \& \ S[b,c] \ \& \ S[c,\emptyset]$$

$$= \langle \text{void} \rangle$$



In the graph notation, downward exiting links have the same role as pattern-variables. They stand in place of any graph structure. A downward link may actually consist of several links, each connecting to a different node-object. As is the case with iconic notations, labeling of links and nodes is not necessary, the geometric structure of the graph is sufficient to identify valid rule applications.

The graph representation of Replication incorporates structure sharing. Rather than replicating the graph structure of Object-a in its entirety, only the links connecting to Object-a are replicated. The rule of Replication shows one unique node with two referential links; in contrast, the rule of Calling replicates two identical nodes. Structure sharing distinguishes object/node replication from reference/link replication. As discussed later in Section 9.3, symbolic notation is not sufficiently expressive to accommodate general structure sharing. And as discussed in Section 9.5.2, the representation of multiple objects by multiple nodes is itself notation dependent.

### 7.1.2 Initials of the Algebra

The initials of the algebra of LoF are next. As Spencer Brown points out, the Position rule could have been stated in a stronger form, as  $(a \ b \ (a)) = \ .$

POSITION

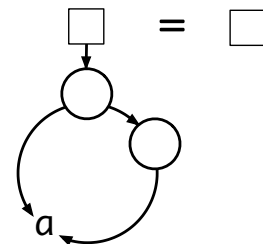
$$(a \ (a)) = \quad \quad \quad (xu_-(y_-(y_-)_c)b)_u = (xu_-)_u$$

$$\{(u,b),(b,y_-),(b,c),(c,y_-)\}$$

$$= \{ \quad \quad \quad \}$$

$$\forall u \forall b \forall c \ S[u,b] \ \& \ S[b,y_-] \ \& \ S[b,c] \ \& \ S[c,y_-]$$

$$= \langle \text{void} \rangle$$



In Position, all downward links that converge upon one location must lead to the same nodes. As representations of pattern-variables, the two links in the graph

of Position may stand in place of many structural links, since the Variable- $a$  is a subgraph label. Without this complete matching, the Position rule takes the shape of the Pervasion rule below. Position is thus a special case of Pervasion.

TRANSPOSITION ◆

$$((a \ b)(a \ c)) = a \ ((b)(c))$$

$$(xu\_(\emptyset \ (y\_ \ z\_))_e(y\_ \ w\_))_f)_d)_u = (xu\_ \ y\_(\emptyset \ (z\_))_e(w\_))_f)_d)_u$$

$$\begin{aligned} & \{ \quad \quad \quad (u,d), (d,\emptyset), (d,e), (d,f), (e,y\_), (e,z\_), (f,y\_), (f,w\_)\} \\ = & \{(u,y\_), (u,d), (d,\emptyset), (d,e), (d,f), \quad \quad \quad (e,z\_), \quad \quad \quad (f,w\_)\} \end{aligned}$$

$$\begin{aligned} \forall u \forall d \forall e \forall f \quad & S[u,d] \ \& \ S[d,\emptyset] \ \& \ S[d,e] \ \& \ S[d,f] \ \& \ S[e,y\_]\ \& \ S[e,z\_]\ \& \ S[f,y\_]\ \& \ S[f,w\_]\ \\ = & S[u,y\_]\ \& \ S[u,d] \ \& \ S[d,\emptyset] \ \& \ S[d,e] \ \& \ S[d,f] \quad \quad \quad \& \ S[e,z\_]\quad \quad \quad \& \ S[f,w\_]\end{aligned}$$

◆

Instead of deletion of irrelevant structure, Transposition rearranges structure. In all notational styles, the descriptive complexity of rearrangement, as opposed to deletion, is apparent. Transposition can be generalized across breadth. The rule applies regardless of the cardinality of replications. In parens notation, generalized Transposition might be written with a continuation,

$$\text{GENERALIZED TRANSPOSITION} \quad a \ ((b)(c)(d)\dots) = ((a \ b)(a \ c)(a \ d)\dots)$$

### 7.1.3 Computational Initials of the Algebra

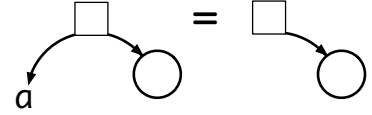
The final two transformation rules, together with Involution (above), are a computational basis for the algebra of LoF. These three void-based rules simplify arrangements solely by deletion of structure. In the LoF text, Dominion is named Integration, Involution is Reflexion, and Pervasion is Generation. The different names are introduced in order to emphasize that this basis is a different ground upon which to build the concepts of the algebra of LoF.

DOMINION ◆

$$a \ ( \ ) = ( \ ) \quad \quad \quad (x\_(\emptyset)_b)_u = ((\emptyset)_b)_u$$

$$\begin{aligned} & \{(u, x_-), (u, b), (b, \emptyset)\} \\ = & \{(u, b), (b, \emptyset)\} \end{aligned}$$

$$\begin{aligned} \forall u \forall b \quad & S[u, x_-] \& S[u, b] \& S[b, \emptyset] \\ = & S[u, b] \& S[b, \emptyset] \end{aligned}$$



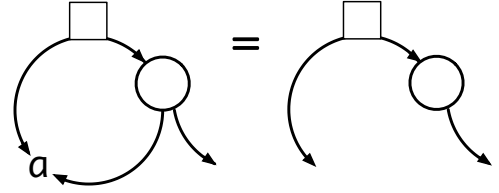
PERVASION

$$a \ (a \ b) = a \ (b)$$

$$(x_- \ y_- (y_- \ z_-)_c)_u = (x_- \ y_- (z_-)_c)_u$$

$$\begin{aligned} & \{(u, y_-), (u, c), (c, y_-), (c, z_-)\} \\ = & \{(u, y_-), (u, c), \quad \quad (c, z_-)\} \end{aligned}$$

$$\begin{aligned} \forall u \forall c \quad & S[u, y_-] \& S[u, c] \& S[c, y_-] \& S[c, z_-] \\ = & S[u, y_-] \& S[u, c] \quad \quad \& S[c, z_-] \end{aligned}$$



## 7.2 Properties of the Iconic System

Using relatively conventional techniques, Spencer Brown provides proof of the independence of the initials of the LoF arithmetic and algebra, as well as uniqueness and convergence of simplification, and soundness and completeness of the algebra. To address these properties of the algebraic system, we adopt a different approach than Spencer Brown. We will use the three computational axioms of the LoF algebra rather than Spencer Brown's two algebraic theorems, since all three are void-based. The central idea is the Principle of Void-Equivalence:

Structures that are equivalent to void are semantically inert and syntactically irrelevant.

The Principle asserts that the background upon which arrangements are recorded does not differentially interact with the interpretation of these arrangements.

The surface of recording does make a difference. In non-Euclidean geometry, for instance, the sum of the interior angles of a triangle differs when the triangle is drawn upon a sphere rather than a plane. However, the impact of the background is homogeneous, different rules do not apply to different triangles drawn on the same background. Homogeneity is a necessary consequence of a structureless and non-addressable substrate, basically because there is nothing to homogenize. Similarly, the meaning of a symbol does not change when the background it is recorded upon does not change.

The key perspective is that absence, regardless of whether or not that absence has an illusory structural representation, can have no impact on the meaning of an existent form. In the LoF iconic system, reduction is just the elimination of void-equivalent structure.

In the arithmetic of containers, arrangements reduce to one of two equivalence classes. The simple values that define these equivalence classes are mark and void. Void-equivalence says that every arrangement in the equivalence class that reduces to void is an irrelevant arrangement, and thus does not impact semantics. When all void-equivalent structure is removed from an arrangement, only a mark will remain, although it is possible that nothing remains. The immediate implication is that void-based reduction is syntactically both consistent and complete. Since parens arrangements are the semantic domain, void-based reduction is also semantically both sound and complete.

### 7.2.1 General Recursive Template

Containment arrangements, as constructed, incorporate a general decomposition strategy. Consider the definition of an arrangement,

Arrangement: (x\_) is an arrangement,  
where x\_ stands in place of zero, one or many other arrangements.

When x\_ is bound to zero objects, the empty container, ( ), provides a ground case. When x\_ is bound to one object, the arrangement will have the form ((a b ...)), where a, b,... are the contents of the single contained object. When x\_ is bound to more than one object, the arrangement will have the form (a b ...). In both latter cases, the separate objects a, b,... are independent, and thus can be safely decomposed recursively in parallel. The value of an arbitrary arrangement then can be determined by the values of the component arrangements separately. Any arrangement found to be void-equivalent is immediately deleted (operationally, the process of finding an arrangement to be void-equivalent is the same process as deleting it). Any discovered empty container immediately triggers Occlusion, which deletes the parent container and all its contents.

### 7.2.2 Articulated Definition

Let the set of valid containment arrangements be A. Here is an alternative construction of the set A: A is the union of two non-intersecting sets N and M. N and M are constructed by mutual recursion. This definition uses conventional variables that bind to one arrangement only. Side-by-side structures (which are still contained by a universal container) are permitted.

$( ) \in M.$   
 $(( )) \in N.$

If  $x \in M$  then  $(x) \in N.$   
 If  $x \in N$  then  $(x) \in M.$

If  $x \in M$  and  $y \in A$ , then  $x y \in M.$   
 If  $x \in N$  and  $y \in M$ , then  $x y \in M.$   
 If  $x \in N$  and  $y \in N$ , then  $x y \in N.$

The above definition can be seen to be an articulation of the conventional definition of valid parenthesis forms,

$( )$  is well-formed.  
 If  $x$  is well-formed so is  $(x).$   
 If  $x$  and  $y$  are well-formed, so is  $x y.$

The two sets  $N$  and  $M$  are the two equivalence classes created by applying the arithmetic of LoF to parens forms. The definition consists of two ground objects, two bounding recursions that implement Crossing, and three multiple contents recursions that implement Calling.

The articulated definition is both a syntactic specification of well-formed structures (i.e. valid parens arrangements), and a semantic specification of the equivalence classes of these structures (i.e. the simple arrangements to which any constructed structure reduces). The interpretation function that determines semantic intent and the construction function that determines structural validity are the same function.

### 7.2.3 Arithmetic

The arithmetic of LoF addresses arrangements without variables. The two rules, Crossing and Calling, can both be read as assertions of void-equivalence.

CROSSING             $(( )) =$

An empty double-container is void-equivalent.

CALLING             $( ) ( ) = ( )$

A replicated empty container is void-equivalent.

By the Principle of Void-Equivalence, both the empty double-container and the replicated empty container do not impact the value or the interpretation of an arrangement.

The two rules can also be stated in the annotated parens format,

$$\text{CROSSING} \quad (xu_{-}(\emptyset (\emptyset)_c)_b)_u = (xu_{-})_u$$

An empty double-container can be added or deleted anywhere.

$$\text{CALLING} \quad (xu_{-}(\emptyset)_a(\emptyset)_a)_u = (xu_{-}(\emptyset)_a)_u$$

A replicated empty container can be added or deleted from any contents.

Crossing and Calling are independent because they address structurally independent arrangements. That is, double-containers and replicated containers are non-comparable structures. Convergence for a void-based substitution system is trivial, any structure that can be deleted should be deleted. Since the Arithmetic rules identify void-equivalent arrangements, their application cannot change the value or the equivalence class of the form being reduced; that is, reduction is consistent.

Completeness asserts that there is no arrangement that cannot be reduced by the two rules. Using the articulated definition above, valid parens forms are co-defined with their reductions. Completeness (as well as other system properties) is true by construction.

## 7.2.4 Algebra

The same general principles apply for the algebra of LoF. Spencer Brown provides proof of independence, consistency, and completeness of the algebra by induction over a canonical form of arrangements.

Algebraic properties can be deferred to the arithmetic. Any variable in the algebraic rules stands in place of zero, one or many arithmetic arrangements, so that the algebraic rules are merely a shorthand notation for arrangements in the arithmetic. However, it is desirable to treat the algebra independently of the arithmetic, permitting variables to stand in place of arbitrary objects that themselves participate in substitution and reduction.

Here, the algebraic properties of the computational axioms are addressed from the perspective of void-equivalence.

$$\text{DOMINION} \quad a ( ) = ( )$$

Arrangements in the presence of an empty container are void-equivalent.

$$\text{INVOLUTION} \quad ((a)) = a$$

Double-containers are void-equivalent.

$$\text{PERVASION} \quad a (a \ b) = a (b)$$

Replicated arrangements in the nested scope of an original are void-equivalent. Stated in terms of annotated parens notation, the three computational rules are

$$\text{DOMINION} \quad (x\_(\emptyset)_b)_u = ((\emptyset)_b)_u$$

When an object that contains an empty container, any other contents can be added or deleted.

$$\text{INVOLUTION} \quad (xu\_(\emptyset \ (y\_)_c)_b)_u = (xu\_ \ y\_)_u$$

Double-containers can be added or deleted anywhere.

$$\text{PERVASION} \quad (x\_ \ y\_ (y\_ \ z\_)_c)_u = (x\_ \ y\_ (z\_)_c)_u$$

Replicated arrangements in the nested scope of an original can be added or deleted.

Independence of the computational algebraic initials is by structural inspection. Dominion applies only in the presence of an empty container. Involution applies only in the case of the double-container. Pervasion applies only to replicated arrangements. No initial can be demonstrated from the others since the structural patterns that each addresses are themselves independent.

Consistency again rests upon the Principle of Void-Equivalence. Application of a rule that deletes void-equivalent structure cannot change the equivalence class of an arrangement. That is, a sequence of steps that would result in changing the value of an arrangement from marked to void would require, at some step, that the mark is changed into a void. However, none of the three axioms change a mark (or any other structure that is not void-equivalent).

The three rules are consistent because they delete only void-equivalent structures. Since the arrangements transformed by these rules are from the semantic domain, the algebra is sound. Since no arrangements change equivalence classes, those that are in the M equivalence class and those that reduce to ( ) are, by construction, the same set. The algebra is thus both syntactically and semantically complete.



### 7.2.5 Interpretation

We have taken non-overlapping containment arrangements to be the domain of discourse. This conceptualization of the world includes a collection of flexibly sized containers and an ability to place containers within containers. The iconic representation illustrates the conceptualization, the predicate calculus representation describes the conceptualization. The conceptualization also includes the LoF calculus which permits adding and removing containers when certain specific patterns of containment are present.

An interpretation is a one-to-one mapping between elements in a representation and elements in a conceptualization. In the iconic parens notation, each parens maps directly onto a container, and the containment relations of parens objects map onto the same containment relations of container objects. In the symbolic relational notation, labeled objects map onto containers, and relational pairs map onto containment structures.

The concept of satisfiability is the linchpin between symbolic representation and actuality. Satisfiability relativizes truth, a symbolic formula is TRUE whenever the symbolic forms can be interpreted as mapping to an actual arrangement. A symbolic formula is valid if it is satisfied by every interpretation and by every variable assignment.

But from the perspective of void-equivalence, a valid formula is one in which all variables are void-equivalent. Iconic notation does not need to introduce truth or logical connectives or interpretative mappings. An iconic form is actual whenever it exists. "Non-satisfaction", i.e symbolically FALSE formulas, are relegated to the iconic void. That is, contradictory actual arrangements cannot be represented by the iconic notation. Variable satisfiability is not a relevant concept because iconic pattern-variables are always satisfied.

Symbolic and iconic interpretations are both ambiguous in that both symbols and icons can be mapped to different conceptualizations. No interpretation is unique. However, there is a substantive difference: iconic representations are the interpretations, while symbolic representations are arbitrary. Different interpretations can be accessed by geometric and topological transformation of iconic representations, while symbolic representation is intentionally independent of any interpretation. Separation of representation and meaning permits symbolic structures to represent purer abstractions; combination of representation and meaning permits iconic structures to represent a more general concreteness.

## 7.2.6 Container Semantics

The semantic domain of LoF is arrangements of containers. The set of possible arrangements,  $A$ , can be defined abstractly as the set of rooted trees. The root of each tree is a single outermost container,  $[ ]$ . Empty containers,  $( )$ , are the leaves of each tree. Ordering of branches is irrelevant. There are no cyclic loops. Rooted trees then are iconic abstractions of realizable physical containment arrangements.

Parens notation represents container arrangements (and rooted trees) as well-formed arrangements of parentheses. Explicit emptiness is denoted by a null token,  $\emptyset$ , when necessary. The mapping from rooted trees to parens forms is many-to-one since parens ordering is incidental. For example,  $(( )( ( )))$  and  $(( ( ))( ))$  are not different arrangements.

Pure containment requires that no container boundaries be breached. This constraint renders the concept of intersection irrelevant. Each boundary is uninterrupted. Abstract containment requires that all physical characteristics associated with the concept of a container be removed. In practice, this means ignoring size, shape, weight, even dimension. An abstract container does only one thing: contain any object. The empty container contains no object. Only the outermost container contains every object. And most importantly, the void within a container, and between multiple containers, is forbidden to participate in any aspect of the abstract model. From a graph representation, this means that links are only Contains relations, there are no links to objects that do not exist, and there are no links between children nodes.

The label of a container also identifies the collection of objects it contains, each object with its own label and its own contents. By incorporating labels of sub-arrangements, the LoF algebra can address containment patterns more efficiently. For example

$$\text{let } x = (( )( )) \quad ( (( )( )) ((( )( ))) ) \Rightarrow (x (x))$$

Identical labels identify identical patterns, but not the same physical containers. However, LoF pattern-variables are more general, they can also identify arbitrary patterns. That is, in the abstract arrangement  $(x (x))$ , Variable- $x$  stands in place of any arrangement at all.

The computational rules define certain realizable patterns to be void-equivalent. The assignment of which patterns are to be considered as irrelevant determines additional semantic domains. The rules can also be motivated as techniques to eliminate cyclic structures, i.e. to eliminate non-realizable containment arrangements. The difference in usage is the distinction between labels and variables. In particular, multiple references to the same label are multiple references to the same container. Multiple references to the same

variable identify the same abstract sub-arrangement in different locations within a given arrangement. Thus, for example, (1 (1)) represents a violation of the Physicality Constraint that no object can be inside two different containers, while (a (a)) describes an arrangement with identical sub-arrangements constituted out of different containers.

The Position rule excludes self-containment by asserting that the arrangement of a container within itself is void-equivalent, that is, (1 (1)) is not a valid container arrangement. However, there is no semantic reason that (a (a)) should be void-equivalent. This question also occurs in the label-free arithmetic of LoF. Removing the references from (1 (1)) results in an empty double container, (( )). The arithmetic interprets this arrangement cyclically, as a specific object, ( ), inserted into itself. (( )) is thus void-equivalent.

$$\text{Label-free Position:} \quad [(\emptyset (\emptyset))] = [\emptyset]$$

The arithmetic also treats replication of labels in the same manner. In the arrangement (1 1) for example, one of the instances of Container-1 is void-equivalent, since physical objects do not have identical twins. Without labels, the Calling rule also suppresses replication of the empty container.

$$\text{Label-free Replication:} \quad [(\emptyset)(\emptyset)] = [(\emptyset)]$$

Both of these rules can be interpreted as suppression of non-realizable arrangements due to violation of the uniqueness of physical objects. In the arithmetic, there is only one container, ( ), that is replicated and self-inserted to construct arrangements of the container with itself. Crossing and Calling reestablish the uniqueness of the single container. Thus, the two equivalence classes created by the arithmetic rules have a semantic interpretation as realizable container configurations. The algebra of labels (not of variables) has the same semantic intent, to eliminate arrangements with multiple reference to the same object.

Since each container itself is an object, there are two types of configurations of one single object, extended replication and extended self-insertion,

$$\begin{array}{ll} \text{Object Replication:} & 1 \ 1 \ 1 \ \dots \\ \text{Object Self-insertion:} & (((1)_1)_1 \ \dots)_1 \end{array}$$

Without suppression, these structures are infinite, and can be interpreted as ways to generate cardinal and ordinal numbers. The semantic constraint of physical uniqueness suppresses replication, and creates an oscillation in self-insertion,

$$\text{Suppressed Replication:} \quad (\emptyset)_1 \ (\emptyset)_1 = (\emptyset)_1 = 1$$

$$\begin{aligned}\text{Suppressed Self-insertion: } & ((\emptyset)_1)_1 = (1)_1 = \emptyset \\ & ((1)_1)_1 = (\emptyset)_1 = 1\end{aligned}$$

The arithmetic of LoF can then be interpreted purely as enforcement of the physical uniqueness of containers. However, the choices to delete replicated objects one at a time, and to delete self-inserted objects two at a time are arbitrary. These choices then reflect a different conceptualization of containment. This additional mechanism allows the LoF arithmetic to map to logic.

It may be that Spencer Brown chose this mechanism explicitly to map to logic. Regardless of motivation, what is of interest is that this mechanism is sufficient to define elementary logic. Logical form is the form of containment arrangements; logical deduction is suppression of cyclic containment by the specific techniques of eliminating replicates and deleting self-containments. The interpretation of LoF as logic is discussed in detail in Section 10.

The two mechanisms that suppress cyclic containment arrangements can be generalized to any object configuration, not just to the empty container object. Here the semantic intent is that variable labels identify multiple reference to the specifically the same object.

$$\begin{aligned}\text{No Object Replication: } & \text{object object} = \text{object} \\ \text{No Object Self-insertion: } & (\text{object})_{\text{object}} = \emptyset\end{aligned}$$

In the case of self-insertion, an object can be identified by its contents, permitting a slightly different representation.

$$\begin{aligned}\text{Object and Contents: } & (\text{contents})_{\text{object}} \\ \text{Object Self-insertion: } & (\text{contents } (\text{contents})_{\text{object}})_{\text{object}}\end{aligned}$$

The Pervasion rule excludes cyclic containment by removing self-inserted contents, thus changing the self-inserted object into a different object with different contents,

$$\text{Pervasion: } (\text{contents } (\text{contents})_{\text{object}})_{\text{object}} = (\text{contents } ( )_{\text{object}}')_{\text{object}}$$

This technique must have the same consequence, the Dominion rule assures that it does,

$$\text{Dominion: } (\text{contents } ( )_{\text{object}}')_{\text{object}} = (( )_{\text{object}}')_{\text{object}'} = \emptyset$$

With all contents removed, the original empty object self-insertion is obtained.

Container semantics is insufficient to construct a Boolean algebra. One other algebraic rule, Involution, is needed. Involution is not well motivated from a container semantics point of view.

Involution:  $(\emptyset (\text{contents1})_{\text{object1}})_{\text{object2}} = \text{contents1}$

Involution is the first rule to incorporate more than one object, and can be interpreted as addressing, via the double-container, explicitly unnecessary containment. That is, object1 is sufficient to contain its contents, object2 does not contribute to that containment.

## 7.2.6 Universe Semantics NOPE

## 7.3 Rule Application by Pattern-Matching

LoF arrangements are spatial containment patterns. LoF transformation rules can be implemented by pattern-matching. Pattern-matching is also a convenient way to implement descriptive changes in sets of ordered pairs. These implementation details are not usually of concern to a mathematical description. A thesis is that these details are in fact essential for an accurate description, particularly when translating between spatial patterns and symbolic representations of these patterns.

Rules are applied by matching the syntactic structure on one side of a rule, and then substituting the other side. This approach permits the general pattern-matching function of unification to identify potential transformations of a given arrangement. When labels align, a symbolic transformation is valid. When patterns align, an iconic transformation is valid.

The presence of  $\emptyset$  causes a rule to fail whenever it is matched by a label other than itself. This is equivalent to asserting that a particular relation cannot exist in a descriptive set for the rule to apply. The absolute use of  $\emptyset$  to indicate an empty container requires an exact match of  $\emptyset$ . The relative use of  $\emptyset$  to mean "no others" requires that ordered pairs such as  $(a, \emptyset)$  fail to match after other explicit pairs have been matched. Of course, the null token is not part of an iconic representation since absence is represented by empty space.

Table 7.2 shows an example of rule application by pattern-matching. Ordered pairs that contain pattern-variables are repeated as often as needed within the symbolic form, so that the pattern-variable can bind to multiple objects. Symbolic variables are existentially quantified; a match will not occur unless a structure exists to create the match.

---

PERVASION	$a (a \ b) = a (b)$	$(x\_ \ y\_ (y\_ \ z\_))_c)_u = (x\_ \ y\_ (z\_))_c)_u$
	$\{(u,y\_),(u,c),(c,y\_),(c,z\_)\} = \{(u,y\_),(u,c),$	$(c,z\_)\}$
Target arrangement:	$[ \ 2 \ (2 \ 3 \ 4)_1 \ ]$	
Rule application:	$2 \ (2 \ 3 \ 4)$ $a \ (a \ b \ )$ $2 \ ( \ 3 \ 4)$	initial arrangement match $a/2,b/3 \ 4$ apply rule
Ordered pairs:	$\{(U,2),(U,1),(1,2),(1,3),(1,4)\}$ $(u,y\_)(u,c) \ (c,y\_)(c,z\_)(c,z\_)$ $\{(U,2),(U,1), \quad (1,3),(1,4)\}$	initial arrangement match $u/U,c/1,y\_/2,z\_/3 \ 4$ apply rule
Result arrangement:	$[ \ 2 \ ( \ 3 \ 4)_1 \ ]$	

---

Table 7.2: An Example of Rule Application

Pattern-variables in the rule are matched to the concrete numeral labels of the target arrangement. The pattern-matching regime consists of these rules,

- If any ordered pair in the rule does not match, the rule application fails.
- If any ordered pair in the target arrangement fails to match the rule, that pair is treated as incidental and simply remains in the descriptive set.
- Pattern-variables in any rule always succeed in matching.
- Matching the domain of an empty container after all explicit pairs are matched causes the rule to fail.
- If the rule succeeds by matching all of its component pairs, then the matched pairs are replaced by pairs on the other side of the rule, with the matched concrete labels substituted for the rule's pattern-variables.

## 7.4 Single Variable Calculus

The LoF computational algebra is a collection of three equations that assert void-equivalence. Below, these rules are first stated using variables to explicitly identify structure that is incidental to each rule. Then we show a

notation that makes incidental structure implicit, and reduces each rule to one variable only.

The Pervasion rule asserts that any replicated labels deeper than a given label are void-equivalent. Variable-b and variable-c identify incidental structure.

$$\text{PERVASION} \quad [a (a \ c) \ b] = [a (c) \ b]$$

With the explicit incidental variable convention, Involution has the form

$$\text{INVOLUTION} \quad [(\emptyset (a)) \ b] = [a \ b]$$

Finally, Dominion does not require incidental variables

$$\text{DOMINION} \quad [a ( \ )] = [( \ )]$$

The incidental variables are reminders that anything may also be present in any given container. This idea can be stated negatively using  $\emptyset$  to signify that nothing else can be present, resulting in one variable rules. Below, each rule has also been generalized to be context-free.

$$\begin{array}{ll} \text{One Variable Dominion:} & a (\emptyset) = (\emptyset) \\ \text{One Variable Involution:} & (\emptyset (a)) = a \\ \text{One Variable Pervasion:} & a (a) = a ( \ ) \end{array}$$

Any container that does not include a null token may contain any other contents. For example, the "( )" in Pervasion might have contents.

Three one variable equations then specify the computational basis rules of LoF. Table 7.3 shows the one variable representation of the three rules, and the corresponding ordered pairs representation.

RULE	ICONIC/ANNOTATED	ORDERED PAIRS
DOMINION	$a (\emptyset) = (\emptyset)$ $(a_-(\emptyset)_b)_u = ((\emptyset)_b)_u$	$\{(u, a_-), (u, b), (b, \emptyset)\}$ $= \{ \quad (u, b), (b, \emptyset) \}$
INVOLUTION	$(\emptyset (a)) = a$ $(xu_-(\emptyset (a_-)_c)_b)_u = (xu_- a_-)_u$	$\{(u, b), (b, \emptyset), (b, c), (c, a_-)\}$ $= \{ \quad (u, a_-) \}$
PERVASION	$a (a) = a ( \ )$ $(x_- a_-(a_- z_-)_c)_u = (x_- a_-(z_-)_c)_u$	$\{(u, a_-), (u, c), (c, a_-)\}$ $= \{(u, a_-), (u, c), \quad \}$

Table 7.3: LoF Single Variable Calculus

The ordered pairs deletion rules are largely unmodified. Pervasion has the only change, one pair was eliminated by treating it as incidental structure. The pattern-variables in the rules have all been renamed  $a_$ , to emphasize the single variable structure. Participating parens are labeled "u", "b" and "c".

The one variable calculus treats non-participating structure as truly non-participating, even in the notation. From the iconic perspective, the patterns created by labeled parens are structural constants that do not require labeling. The null token is also a constant. The significance of this formulation is that no other mathematical theory uses only one variable.

## 8 Abstract Algebra

The formal description of LoF includes the theory of equality relations (Section 5.2), which then leads to characterizing LoF as an abstract algebra. A simple algebraic structure consists of a set and one or more operations on elements of that set that satisfy specific axioms. Algebras do not usually incorporate relations. Here, the Contains relation is converted into the PUT function, which then permits LoF to be analyzed as an abstract algebra. Kohout and Pinkava's [ref] mischaracterization of LoF as a Boolean algebra, and Meguire's mischaracterization as a lattice [ref] are both discussed in Section 10.

PUT forms a non-associative quasigroup over parens arrangements. The LoF rules further restrict the PUT function to an algebraic structure that little is known about.

### 8.1 The PUT Function

The parens language represents all valid containment arrangements, and is identical to the set of unordered rooted trees (Section 10.4). The set of parens arrangements,  $A$ , is the domain of an algebra of containment. Each member of  $A$  can be described by a set of ordered pairs. Each set of ordered pairs represents the collection of containment formulas that constitute the particular parens arrangement.

The Contains relation can also be portrayed as a binary operation, that of inserting or putting one container-object inside the boundary of another container-object.

Object-b Contains Object-a

PUT Object-a into Object-b



The PUT operation,  $a \oplus b$ , generates arrangements that are described by the Contains relation.

Container-objects can be described in both symbolic and iconic forms,

$$a =_{\text{def}} (xa\_)_a$$

The pattern-variable  $xa\_$  generalizes Object- $a$  to have any contents. The iconic representation provides the meaning of the symbolic representation, which creates an unusually rich interpretive context for the PUT function. Specifically, both the symbolic form and its interpretation can be written side-by-side. What the symbolic form means is then directly visible.

$$a \oplus b \text{ is } S[b,a] \text{ is } (a \text{ } xb\_)_b$$

All parens arrangements can be constructed from one ground object, the empty container  $()_e$ , and the single binary function, PUT. The empty container specifically does not contain a pattern-variable. There is no occasion to require quantification of pattern variables (such as the case of asserting that a container contains at least one object). This is consistent with the algebraic approach, which also does not provide quantification.

The PUT function is closed, since applying PUT to elements of the set of arrangements generates another element in the same set. The algebraic structure of PUT,  $\langle A, \oplus \rangle$ , thus consists of

The grounded set of arrangements (rooted trees),  $A$ .

The closed binary PUT operation,  $a \oplus b$ , such that  $A \times A \rightarrow A$ .

The PUT operation itself can be defined iconically as

$$\text{PUT:} \quad a \oplus b =_{\text{def}} (a \text{ } xb\_)_b$$

No assumptions are made about  $xb\_$ , including the possibility that Object- $a$  is already contained by Object- $b$ .

Every object can be PUT into some other object, satisfying the Existence Constraint of functions. The Uniqueness Constraint of functions is also satisfied, since the result of putting one specific object into another is unique.

## 8.2 Construction of Arrangements

Any particular arrangement can be expressed as a series of PUT operations on labeled containers.  $( )_e$  is named  $e$  in the algebraic form. The double-container expressed as PUT operations is

$$e \oplus e \implies (( ))$$

An empty double-container is generated by putting an empty container into another empty container. Two empty containers as the content of the same object can be expressed as

$$e \oplus . e \oplus e \implies (( )( ))$$

This arrangement is generated by putting another empty container into an empty double container.

An immediate question is whether or not there is only one empty container. Can empty containers be replicated freely; do replications have to be distinguished? Similar to the use of numerals in the algebra of numbers, we will permit free replication, but label different empty containers as  $( )_{en}$  for identification. The subscript "e" identifies the container as empty; the subscript "n" identifies different empty containers with a numerical index. Once an empty container is no longer empty, we will drop the "e" from the label. This provides a natural naming convention for compound arrangements, they are named by their outer container. In any sequence of PUT operations, the right-most container label identifies the entire arrangement. For example, the generation of an empty double-container would be labeled

$$e1 \oplus e2 \implies (( )_{e1})_{e2} \implies (( )_{e1})_2$$

The name of the newly constructed double-container is "2". Obviously, an empty container loses its empty status whenever it is PUT into.

Functions can be nested, functional composition can then be expressed across nested function applications. Let "f" name the PUT function. The double-container above can also be represented as  $f(e, f(e, e))$ . The structure of PUT function nesting does not map onto the structure of the containment arrangement constructed by the function.

### 8.2.1 Functional Construction

Here is an annotated arrangement expressed as a sequence of PUT construction operations,

$$\text{Annotated Parens:} \quad [(\emptyset)_{e1} ((\emptyset)_{e3} ((\emptyset)_{e5})_4)_2]$$

Function Composition:  $e5 \oplus e4 \cdot \oplus \cdot e3 \oplus e2 \text{ :}\oplus\text{ : } e1 \oplus U$

Nested functions:  $f(f(f(e5,e4),f(e3,e2)),f(e1,U))$

This sequence of PUT applications constructs the above annotated arrangement,

$e5 \oplus e4$	$\implies$	$((\ )_{e5})_4$
$e3 \oplus e2$	$\implies$	$((\ )_{e3})_2$
$4 \oplus 2$	$\implies$	$((\ )_{e3} ((\ )_{e5})_4)_2$
$e1 \oplus U$	$\implies$	$[(\ )_{e1}]$
$2 \oplus U$	$\implies$	$[(\ )_{e1} ((\ )_{e3} ((\ )_{e5})_4)_2]$

This arrangement can also be constructed by a different sequence of operations, for example,

$e1 \oplus \text{ : } e3 \oplus \text{ : } e5 \oplus e4 \cdot \oplus \cdot e2 \text{ :}\oplus\text{ : } U$

This can also be expressed as nested functions by

$f(e1,f(f(e3,f(f(e5,e4),e2),U))$

The different construction sequences are specified by one transformation that expresses the idea that the contents of any container are mutually independent. Independence of container contents is expressed functionally as commutativity over successive compositions.

Temporal Commutativity:  $a \oplus \cdot b \oplus c = b \oplus \cdot a \oplus c$

$f(a,f(b,c)) = f(b,f(a,c))$

Labels can be exchanged across depth of nesting. PUT commutes only with objects that have already been PUT into the common container,  $(xc\_)_c$ , however objects do not commute with the common container itself.

$a \oplus (b \ xc\_)_c = b \oplus (a \ xc\_)_c$                        $(a \ b \ xc\_)_c = (b \ a \ xc\_)_c$

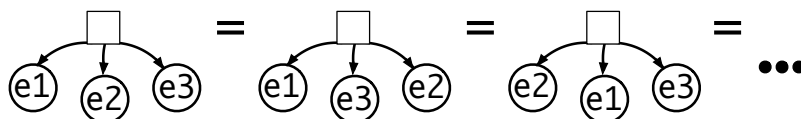
$a \oplus (xc\_)_c \neq (xc\_)_c \oplus a$                        $(a \ xc\_)_c \neq (c \ xa\_)_a$

Atemporally, commuting across container boundaries is operational parallelism. The sequence of composition is irrelevant when all objects are PUT into a container concurrently. Arity is not necessarily a property of parallel operators because operations are applied to all arguments concurrently. Thus

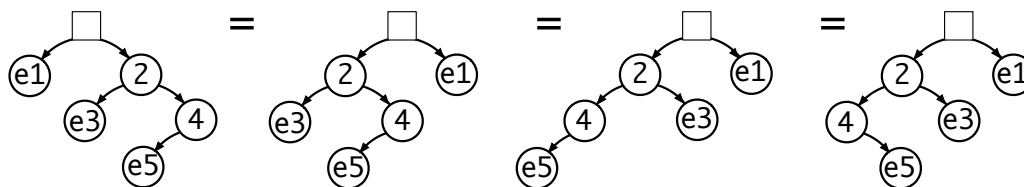
the commutative and associative properties of binary functions are induced by a sequential representation that includes time structurally but not conceptually.

## 8.2.2 Notational Flexibility

The algebraic rules of the PUT function identify equivalent arrangements that are generated both by different arguments to a single application of the function (e.g. commutativity, idempotency), and by different compositions of function applications (e.g. associativity). Arrangements are unordered rooted trees. The iconic structural rule is that the ordering of children nodes is irrelevant. That is,



The tree display is a iconic form, invariant under rotation and translation in three-dimensions. Thus, what is called commutativity in symbolic form is simply rotation in iconic form.



Rotation, in turn, can be seen as moving the viewpoint of observation. Symbolic notation does not include the viewpoint of the reader, so commutativity must be expressed as a structural transformation of the representation. Iconic notation, in contrast, incorporates the concept of viewpoint, so that commutativity does not require restructuring of the representational form, but rather simply looking at the form from a different perspective.

### 8.2.2.1 Commutativity

By abstracting the variety of rotations and traversals of the graph into the singular idea of viewing a spatial object, the behavior of the representation changes from string-like to icon-like. Alternative descriptions of different iconic forms depend upon the dimension of representation. What in a string description is commutativity, in string traversal is sequence of reading. When commutativity is taken as a sequence of visitations, textual layout becomes differentiated not by placement of objects, but by the temporal ordering of reading those objects. When the commutative operator is taken to be a parallel

operation, ordering of arguments is not a relevant property, all arguments are processed, traversed, or viewed at the same time.

In the freedom of three dimensions, spatial transformation can be achieved by a greater diversity of methods. What in two-dimensional graph layout is mirroring, in graph traversal is a choice of which link to visit next. A two-dimensional mirroring is also a three-dimensional rotation. By permitting the reader to inhabit the three-dimensional space of representation, rotation can be expressed equivalently as the movement of the viewpoint of the reader. For example, the first and last figures above express "looking from the other side", there is no modification of the representation, just a modification of the location from which the reading is made. The interpretation of iconic notations, due primarily to the inside/outside orientation of containment boundaries, incorporates the position of the reader. Reading perspective is synonymous to parallel processing.

Figure 8.1 shows examples of the representation of commutativity across different dimensions.

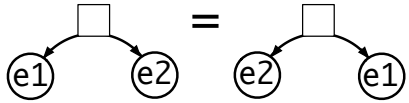
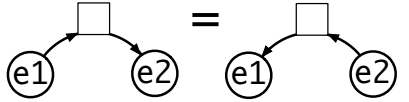
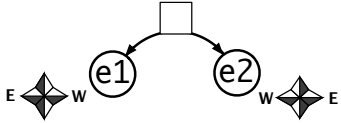
REPRESENTATION	DIMENSION	TRANSFORMATION	EXAMPLE
string	one spatial	commutativity	$e1\ e2 = e2\ e1$
string traversal	one spatial one temporal	reading direction	$e1 \rightarrow e2 = e1 \leftarrow e2$
graph	two spatial	rotation	
graph traversal	two spatial one temporal	visitation sequence	
icon/reader	three spatial	change viewing perspective	

Figure 8.1: Commutativity -- Flexibility of Notation

If meaning is indeed independent of representation, then each of the transformations in Figure 8.1 can be interpreted as conveying the same abstract concept. Symbols embody sequence. A commutative string function is a recording in different sequential locations, it is also a temporal sequence of reading.

Icons embody concurrency. A rotated spatial function is also a temporal visitation sequence. In three-dimensions, the abstract concept of commutativity is a subjective choice of reading position. The point is that the meaning of an abstract symmetry is better conveyed by a diversity of representational strategies, none of which is inherently more or less formal than the other.

The PUT function is not commutative. Changing the ordering of arguments to PUT changes its meaning. Changing the order of reading, rotating the graph representation, changing the visitation sequence of nodes, and viewing from a different perspective would also change the meaning of the function.

Spencer Brown's approach to commutativity is more abstract. The PUT function introduces non-commutativity through its binary structure. Commutativity is within the nature of functions, induced by the concept of a specific arity. However, abandoning functions, from the perspective of iconic patterns, the relation Contains does not introduce the concept of commutativity. Commutativity is not just irrelevant to the iconic notation, it is foreign. This idea returns to a theme, that meaning is not independent of notation. The very concepts that have been used to define modern algebra are string-based concepts, they are representational artifacts when formal ideas are permitted the freedom of a spatial representation. In this sense, representation is not independent of meaning.

8.2.2.2 Associativity

Associativity can also be represented across dimensions. The graph distinction that expresses associativity is over graph depth rather than breadth (as is the case of commutativity). Although depth and breadth are different graph structures, the graph itself unifies these apparently different properties of functions. Figure 8.2 shows the notational flexibility inherent in the associative property.

---

REPRESENTATION	DIMENSION	TRANSFORMATION	EXAMPLE
----------------	-----------	----------------	---------

string	one spatial	commutativity	$e1\ e2 \cdot e3 = e1 \cdot e2\ e3$
string traversal	one spatial one temporal	accumulation sequence	$e1\ e2 \rightarrow e3 = e1 \leftarrow e2\ e3$
graphs	two spatial	depth	
graph traversal	two spatial one temporal	visitation sequence	
graph/reader	three spatial	collapse depth	

Figure 8.2: Associativity -- Flexibility of Notation

Like commutativity, the concept of associativity can be represented in a variety of dimensional forms. Like commutativity, it is also induced by the imposed concept of binary arity.

An associative string function is also a temporal sequence of accumulation. As a graph, this accumulation can be represented by nodes that create additional depth in the representation. The depth can also be represented across breadth as a composition of concurrent and sequential node visitations.

While commutativity is essentially ordering, associativity is essentially grouping. The three-dimensional change of perspective is to collapse the depth of the graph rather than the breadth, which is equivalent to removing the binary arity of the function. Variary structures to not support a grouping concept.

Spencer Brown's approach to associativity is similar to his approach to commutativity. From the perspective of iconic patterns, grouping the contents of a container (i.e. arity) violates their independence. Since containment is the iconic grouping operation, supporting binary arity would require additional containers, making it impossible to express containment of multiple objects by a single container. Associativity is not just irrelevant to the iconic containment, it is subversive. Here the dependence of concept and notation is strong. From one perspective, associativity cannot even be expressed without superfluous containment; from the other, multiple containment inherently does not support the concept of associativity.

The binary PUT function is neither associative nor commutative. Changing the grouping of arguments to PUT changes its meaning, as does exchanging the objects being PUT and being PUT into. The Contains relation has been constructed so that commutativity and associativity are irrelevant, both concepts are absorbed within the conjunction of relations. This makes it easy to see that these properties of containment are induced by the conversion from relational to functional notation. That is, they are dependent on notation. Table 8.1 illustrates these differences.

Semantics	$(b \ c \ d)_a$	$(b \ (c \ d)_g)_a$
Relations	$S[a,b] \ \& \ S[a,c] \ \& \ S[a,d]$	$S[a,b] \ \& \ S[a,g] \ \& \ S[g,c] \ \& \ S[g,d]$
Ordered Pairs	$\{(a,b),(a,c),(a,d)\}$	$\{(a,b),(a,g),(g,c),(g,d)\}$
Composed Functions	$d \oplus : c \oplus . b \oplus a$	$c \oplus . d \oplus g : \oplus : b \oplus a$
Nested Functions	$f(d, f(c, f(b, a)))$	$f(f(c, f(d, g)), f(b, a))$

Table 8.1: Notational Inducement of Properties

In Table 8.1 the relational description of semantic containment arrangements consists of a conjunction of formulas that relate individual objects. Multiple containment is represented by the positioning of labels across formulas. Implicitly, the AND conjunction exhibits commutativity and associativity, but these properties of the logical connective are not about the semantic domain, they are about the logical domain. The set of ordered pairs removes all notions of ordering or grouping that are residual in the logical connectives. The functional notation, however, imposes an ordering on constructive operations that may then be annulled by a particular property of the function. Non-commutativity and non-associativity characterize the binary PUT function. Semantic concepts associated with containment become operational concepts, confounding a particular method of constructing containment arrangements with the resultant arrangement themselves. The concepts of (non)commutativity and (non)associativity describe the functional notation but not the semantic domain.

### 8.3 The PUT Magma



At this point, PUT operations on parens arrangements constitute a magma,  $\langle A, \oplus \rangle$ , a set  $A$  equipped with a single closed binary operation,  $A \times A \rightarrow A$ . Magmas impose no other axioms on an algebraic structure.

The theory of algebraic structures identifies common axioms, such as associativity and commutativity, to which various algebraic operations conform. From the iconic pattern-matching perspective, these properties are not relevant since they describe functions rather than iconic patterns. They are relevant, however, from the perspective of the abstract algebra of PUT.

In describing the algebraic structure of LoF, it is important to maintain the primary characteristic of void-based iconic representation: the void does not participate within the formal system. This constraint has two consequences,

There is no null object.

Multiple objects contained by a container-object are independent.

The PUT function is non-standard for several reasons. Containment is inherently a relation, so that converting a Contains formula into a PUT operation constructs an unusual type of function. PUT, for example, is neither associative nor commutative. The structure of PUT is also constrained by the void-based semantics mentioned above, and will be further constrained by the arithmetic and algebraic rules of the LoF calculus. Finally, algebraic theories, like logical theories, have been developed with string notation in mind, so that algebraic objects are not usually conceptualized as having an interior.

### 8.3.1 Not Associative, Not Commutative

The PUT function was derived from the Contains relation. The irreflexivity of Contains becomes non-idempotency of PUT and the asymmetry of Contains becomes the non-commutativity of PUT. Since relations cannot be composed, there is no constraint on Contains that models the non-associativity of PUT.

The structural demonstration of non-commutativity of PUT is by observation of the corresponding parens arrangements. From the definition of PUT,

$$a \oplus b \implies (a \times b\_)\_b \qquad \{(b,a)\}$$

The composed functional, nested functional, iconic annotated parens, and relational ordered pairs representations are shown. For commutativity,

$$\text{Not Commutative: } a \oplus b \neq b \oplus a \qquad f(a,b) \neq f(b,a)$$

$$\begin{array}{ll}
a \oplus b \implies (a \text{ xb\_})_b & (b,a) \\
b \oplus a \implies (b \text{ xa\_})_a & (a,b) \\
((\text{xa\_})_a \text{ xb\_})_b \neq ((\text{xb\_})_b \text{ xa\_})_a & (b,a) \neq (a,b)
\end{array}$$

The structural demonstration of the non-associativity of PUT is similar,

$$\begin{array}{ll}
\text{Not Associative: } a \oplus b \text{ .} \oplus c \neq a \oplus \text{ .} b \oplus c & f(f(a,b),c) \neq f(a,f(b,c)) \\
a \oplus b \text{ .} \oplus c \implies ((a \text{ xb\_})_b \text{ xc\_})_c & \{(b,a),(c,b)\} \\
a \oplus \text{ .} b \oplus c \implies (a \text{ b xc\_})_c & \{(b,a),(c,a)\} \\
((a \text{ xb\_})_b \text{ xc\_})_c \neq (a \text{ (xb\_)}_b \text{ xc\_})_c & \{(b,a),(c,b)\} \neq \{(b,a),(c,a)\}
\end{array}$$

### 8.3.2 Non-associative Cancellation

Magmas are usually subdivided into two types, semigroups that are associative, and quasigroups that support cancellation and division but are not associative. The PUT operation is not associative, so it does not form a semigroup. The remaining structures to examine are cancellation magmas and division magmas, both together form a quasigroup.

The cancellation properties of magmas, expressed as PUT operations, are

$$\begin{array}{ll}
\text{Magma Left Cancellation: } a \oplus b = a \oplus c & \text{IMPLIES } b = c \\
\text{Magma Right Cancellation: } b \oplus a = c \oplus a & \text{IMPLIES } b = c
\end{array}$$

PUT does indeed form a cancellation magma,

$$\begin{array}{ll}
\text{Left Cancellation: } (a \text{ xb\_})_b = (a \text{ xc\_})_c & \text{IMPLIES } (\text{xb\_})_b = (\text{xc\_})_c \\
\text{Right Cancellation: } (b \text{ xa\_})_a = (c \text{ xa\_})_a & \text{IMPLIES } b = c
\end{array}$$

In the case of left cancellation, when the same object is taken out of equal containers, what remains is equal. The implication goes the other direction as well, when the same object is PUT into equal containers, they remain equal. Similarly for right cancellation, when two objects are PUT into the same container and the results are equal, then the two objects are equal. This implication is also bidirectional, when two equal objects are PUT into the same container, the result maintains structural equality. The proof comes either from arithmetic substitution or from induction over arrangements.

### 8.3.3 The Problem of Labeling

Labeling does, however, create a false impression of iconic forms. When the equality of two containers and their contents is asserted, the equality also applies to labels. That is, by definition,

$$b = c \text{ means } (xb\_)_b = (xc\_)_c$$

From an iconic perspective, this is a case of mislabeling, since the unlabeled iconic forms are identical by construction. The equality is a structural identity. The intent of the cancellation property is to identify transformations that are not structurally identical, but are semantically equal. Functions generate different names for the same object, and therefore construct semantically equal but syntactic different representations. Thus, it makes sense in symbolic algebra to write, for example,

$$(x\ y\ z)_b = (z\ y\ x)_c \text{ IMPLIES } (y\ z)_b = (z\ y)_c$$

From the perspective of string representation, the different labels of Object-b and Object-c are justified because the two string structures are different. The iconic form, in contrast, does not support linear ordering of contents, which makes Object-b and Object-c identical both before (as antecedent) and after (as consequence) the implication.

That is, each iconic arrangement is unique. There is only one name for, more accurately one illustration of, each arrangement. Although sequential constructions may vary, the variation is characteristic of the symbolic algebraic form, and not of the semantic domain of containment arrangements. The string representation imposes incidental structure that is not originally within the semantic domain of containment arrangements. The string-based technique that is used to suppress this incidental structure is to assert string-based axioms, such as commutativity. The example above is in an iconic language, the ordering of contents is an illusion so to speak. Another way to phrase this is that the iconic form permits freedom of viewing perspective in three dimensions, so that contents can be viewed in any ordering.

The iconic technique is to rectify the mislabeling of identical arrangements (called structure sharing, described in Section 9.3). Cancellation is still an important axiom, but it would be written as an identity,

$$\text{Iconic Left Cancellation: } (a\ xb\_)_b = (a\ xc\_)_c \text{ IMPLIES } b = c$$

$$\text{Iconic Right Cancellation: } (b\ xa\_)_a = (c\ xa\_)_a \text{ IMPLIES } b = c$$

Both right and left cancellation are statements that two identical objects, Object-b and Object-c, have different labels.

The question here is whether or not "b" and "c" are distinguishably (i.e. structurally) different. If the labels identify arrangements without variables, then "b" and "c" are different labels for identical objects. The identical objects have no means by which to support structural difference. Therefore, the two labels constitute a labeling confusion in both versions of cancellation. In philosophical language, not only is the reference of the labels identical, but the sense of the labels is also identical. The labels not only identify the same object, the way in which they identify that object is also the same.

One would expect it to be inappropriate to propagate a labeling confusion through any transformation. When the labeling confusion is rectified, the two cancellation rules become the same. In this sense, the iconic form does not distinguish "left" and "right" versions of cancellation. The right/left distinction is an induced artifact rather than a fundamental qualities of an abstract concept.

Another possibility is that "b" and "c" stand in place of algebraic structures which contain variables. It is possible that the two labels identify different structures, making the assertion of equality semantic rather than structural. The critical distinction is whether or not these variables are iconic pattern-variables, or more standard algebraic variables. If they are pattern-variables, then those variables stand in place of anything at all, and thus do not impart a different structure to the arrangement they participate in. Again, within the iconic form, there would be a mislabeling rather than a structural difference. It is only within a string-based algebraic notation that "right" and "left" versions of cancellation can occur. This might be expected, since the terms right and left are not within the vocabulary of a spatially oriented representation.

#### 8.3.4 Non-associative Division

The division axioms for quasigroups are

Quasigroup Left Division:	$a \oplus b = c$	$b = a \backslash c$	b is unique
Quasigroup Right Division:	$a \oplus b = c$	$a = c / b$	a is unique

Transcribing the division axioms into annotated parens notation provides visual clues as to the interpretation of left- and right-division.

$$\begin{array}{lll}
 a \oplus b = c & (a \text{ } xb\_)_b = c & b = (xb\_)_b = a \backslash (a \text{ } xb\_)_b \\
 & & a = (a \text{ } xb\_)_b / (xb\_)_b
 \end{array}$$

Both forms of division appear to be natural deconstruction operations on arrangements. Left-division is a specific content deletion operation;  $a \setminus (a \times b)_b$  can be read as TAKE Object-a out of Container-b, and keep Container-b. Right-division is a specific content retrieval operation;  $(a \times b)_b / (a \times b)_b$  can be read as GET Object-a out of Container-b and keep Object-a.

Both TAKE and GET identify different ways to decompose and consequently compose Object-c. By substitution,

$$\begin{array}{ll} \text{PUT}[a, \text{TAKE}[a, c]] = c & a \oplus a \setminus c = c \\ \text{PUT}[a, \text{TAKE}[a, (a \times b)_b]] = (a \times b)_b & a \oplus a \setminus (a \times b)_b = (a \times b)_b \\ \text{PUT}[a, (a \times b)_b] = (a \times b)_b & a \oplus (a \times b)_b = (a \times b)_b \end{array}$$

This composition of transformations does not work in the case that Object-c does not originally contain Object-a, however Object-c is defined to contain Object-a by the construction of putting an arbitrary Object-a into an arbitrary Object-b. That is to say, Object-c always contains Object-a.

Similarly, by substitution,

$$\begin{array}{ll} \text{PUT}[\text{GET}[c, b], b] = c & c / b \oplus b = c \\ \text{PUT}[\text{GET}[(a \times b)_b, b], b] = (a \times b)_b & (a \times b)_b / b \oplus b = (a \times b)_b \\ \text{PUT}[a, b] = (a \times b)_b & a \oplus b = (a \times b)_b \end{array}$$

Again the composition would fail if Object-c did not contain Object-a, which it does by construction.

The conclusion is that the PUT function forms a quasigroup. However, the division functions TAKE and GET will play no further role in the analysis.

## 8.4 Properties of the PUT Function

The common properties of functions include associativity, commutativity, identity, inverse, and idempotency. PUT is not associative, and it is not commutative. The functional and annotated parens notations are shown.

Not Commutative:

$$a \oplus b \neq b \oplus a \quad ((x a)_a \times b)_b \neq ((x b)_b \times a)_a$$

Not Associative:

$$a \oplus b \oplus c \neq a \oplus (b \oplus c) \quad ((a \times b)_b \times c)_c \neq (a \times (b \times c)_c)_c$$

The elementary object for the set of parens arrangements is the empty container,  $()_e$ . The empty container is not a zero for PUT.

No Zero:

$$a \oplus e \neq e \oplus a \neq a \qquad ((xa\_)_a)_e \neq (( )_e xa\_)_a \neq (xa\_)_a$$

Without a zero, there is no concept of an inverse object.

Since symbolic forms include representation of nothing (egs. the null string, the empty set, the blank word), a symbolic zero might be "nothing", labeled by  $\epsilon$ . Putting Object-a into nothing yields Object-a, as does putting nothing into Object-a. But this violates the fundamental premise of iconic notation, that empty space does not participate in transformation. For iconic notations, labeling nothing is the error of induced symbolism.

PUT is not idempotent because putting a container inside itself is excluded from the conceptualization of containment. Care is needed when addressing replicated objects. In particular, one empty container can be PUT into another, and in general, a replicate can be put into an original. But putting an object into another changes the receiving object. When an empty container is PUT into a second empty container, the second container is no longer empty. For this reason, replicates are better considered to be new objects, just like expressing the sum of  $3 + 3$  treats the first "3" as different from the second "3". Again, replication of object labels is an aspect of induced symbolism which creates incompatibilities between symbolic and iconic representation.

Since the same arrangements can be constructed by different sequences of PUT operations, the PUT function does exhibit compositional, or temporal, invariants. In particular, the order in which objects are PUT into the same container does not matter. What is important is that compound objects must be assembled prior to putting them into a particular container.

Alternatively, from the perspective of parallelism, all objects to be PUT into a contain can be PUT in at the same time, in parallel. The sequence of PUT operations with regard to the same container is an artifact of the functional notation.

## 8.5 Finite Set Theory and LISP Programming

The PUT function is fundamental to both computer programming languages and to finite set theory. The PUT quasigroup consists of three binary functions, PUT, TAKE, and GET.

Ignoring the sequence of lists for a moment, these three functions are integral to the LISP programming language. LISP is a list processing language that incorporates only one native data structure, the list. The empty list is named NIL, and corresponds to the empty container.

Lists are composed by putting objects into the empty list, an operation named CONS. Lists are deconstructed by taking the first object out of the list, a function named CAR, or more conveniently, FIRST. What remains of the list after the first object is removed is identified by the function CDR, or more conveniently, REST. Given a list named "list", LISP then has this invariant,

$$(\text{CONS} (\text{FIRST list}) (\text{LAST list})) = \text{list}$$

If Object-a were the first item in a LISP list, then this invariant is also expressible as PUT functions.

$$(\text{CONS} (\text{FIRST list}) (\text{LAST list})) \Rightarrow \text{PUT}[\text{GET}[a, \text{list}], \text{TAKE}[a, \text{list}]]$$

The constructor PUT and the two destructors GET and TAKE are very common in data processing implementations. Table 8.2 shows the alignment between the PUT function and lisp data accessors.

PUT FUNCTION	LISP PROGRAMMING	FINITE SETS
( ) <sub>e</sub>	NIL	{ }
(xa <sub>-</sub> ) <sub>a</sub>	a	{xa <sub>-</sub> }
PUT[a,b]	(CONS a b)	{xa <sub>-</sub> } U {xb <sub>-</sub> }
GET[a,b]	(FIRST b)	{xa <sub>-</sub> } ∩ {xb <sub>-</sub> }
TAKE[a,b]	(REST b)	{xb <sub>-</sub> } - {xa <sub>-</sub> }

Table 8.2: PUT, LISP and Sets

As well, containers behave similarly to finite sets. The restrictions of ordering (i.e. the FIRST element) and grouping (i.e. only the FIRST element) imposed upon a LISP list are not incorporated in the concept of a set.

The arguments of the PUT function are restricted to be singular objects; this restriction also is not incorporated in sets. As a consequence, the finite set operations are sequential compositions of PUT function applications.

Recursive definitions and proofs are usually composed of steps, so that a definition of set equality might be

$$A = B \quad \text{IFF} \quad x \in A \text{ AND } x \in B \text{ AND } A - \{x\} = B - \{x\}$$

This would be written in LISP prefix notation as a recursive program on lexically ordered lists

```
(OR (AND (= A NIL) (= B NIL))
    (AND (= (FIRST A) (FIRST B))
          (= (REST A) (REST B)) ))
```

Similarly, the recursive definition of the equality of two parens arrangements can be written as

$$(xa\_)= (xb\_)\quad \text{IFF} \quad \text{GET}[c,a] \text{ AND } \text{GET}[c,b] \text{ AND } \text{TAKE}[c,a] = \text{TAKE}[c,b]$$

## 8.6 Functional LoF

The algebraic theory of quasigroups distinguishes various varieties, based primarily on the presence of additional properties, such as an identity element and an inverse. The PUT function does not qualify as an example of these more articulated quasigroups (it is not a loop, a rack or an order). However, PUT does take on more definition when the rules of LoF are incorporated into its structure.

### 8.6.1 Functional Arithmetic

LoF arithmetic can be expressed operationally using the PUT function. Below, labels in the annotated parens notation have been modified slightly to conform to the functional symbolic notation.

CALLING	$(\ )(\ ) = (\ )$	$(xu_{-}(\emptyset)_{e1}(\emptyset)_{e2})_u = (xu_{-}(\emptyset)_{e1})_u$
	$e \oplus . e \oplus u = e \oplus u$	$f(e, f(e, u)) = f(e, u)$
CROSSING	$((\ )) =$	$(xu_{-}(\emptyset (\emptyset)_{e2})_{e1})_u = (xu_{-})_u$
	$e \oplus e . \oplus u = u$	$f(f(e, e), u) = u$

The composition of PUT functions that generates the LoF arithmetic is not associative. Visually above, the two arithmetic rules are associative variations of the same sequence  $e \oplus e \oplus u$ , yet yield different results. In fact, the rules of the arithmetic can be seen as a statement of non-associativity.



## 8.6.2 Functional Algebra

Spencer-Brown's two rules of the LoF algebra can be expressed as compositions of PUT operations,

$$\text{POSITION} \quad (a \ (a)) = \quad (xu\_ (xa\_ (xa\_ )_a)_a)_u = (xu\_ )_u$$

$$a \oplus a \cdot \oplus u = u \quad f(f(a,a),u) = u$$

The functional perspective shows that Position is a generalization of Crossing. Whenever a replicate of any arrangement is placed within that arrangement, the resulting form is void-equivalent. Given the semantic constraint that a container cannot contain itself, Position asserts that if indeed self-containment does occur, then the result is no arrangement at all. It is possible to PUT a replicate of an object into the original, in effect changing the original. In all such cases, the resulting construction does not correspond to any semantic possibility.

Position (and Crossing) are then methods of strictly enforcing the irreflexive constraint in the relational description. In a void-based computational system, one way of asserting a constraint is to make violation of that constraint void-equivalent. What began as an intuitive notion of the physical constraint that objects cannot self-contain, is violated by a symbolic notation that permits free replication of object labels, and is then corrected by LoF rules that assert self-containing arrangements to be void-equivalent.

Transposition, the only rearrangement rule that has been considered, is expressed by PUT operations as

$$\text{TRANSPPOSITION} \quad ((a \ b)(a \ c)) = a \ ((b)(c)) \\ (xu\_ (\emptyset \ (a \ xb\_ )_b(a \ xc\_ )_c)_d)_u = (xu\_ a \ (\emptyset \ (xb\_ )_b(xc\_ )_c)_d)_u$$

$$a \oplus b : \oplus : a \oplus c \cdot \oplus d \cdot : \oplus u = b \oplus \cdot c \oplus d : \oplus : a \oplus u$$

$$f(f(f(a,b),f(f(a,c),d)),u) = f(f(b,f(c,d)),f(a,u))$$

Transposition is a theorem of the simpler computational rules of LoF. Since theorems can be arbitrarily complex, Transposition does not illustrate a fundamental property of the PUT function.

The functional representation of three simpler computational rules helps to clarify the expression of Spencer Brown's two rules above.

$$\text{OCCLUSION} \quad (a \ ( \ )) = \quad (xu\_ (xv\_ ( \ ))_e)_v)_u = (xu\_ )_u$$

$$e \oplus v . \oplus u = u$$

$$f(f(e,v),u) = u$$

Occlusion asserts that putting  $e$  into any container is equivalent to putting  $e$  into itself.

$$e \oplus v = e \oplus e = \emptyset$$

Dominion occurs only in one case, when an empty container is one of the contents of  $U$ . In all other cases, the empty container will be the contents of some outer container. The presence of the outer container in effect converts Dominion into Occlusion. Therefore, the algebraic rule of Dominion is stated with  $u$  as the outer container, since it may stand in place of the non-reducing container  $U$ .

$$\text{DOMINION} \quad a ( ) = ( )$$

$$(xu\_ a ( )_e)_u = (( )_e)_u$$

$$a \oplus . e \oplus u = e \oplus u$$

$$f(a, f(e, u)) = f(e, u)$$

Involution also requires a refinement.

$$\text{INVOLUTION} \quad ((a)) = a$$

$$(xu\_ ((xa\_)_a)_e)_u = (xu\_ xa\_)_u$$

$$a \oplus e . \oplus u = xa\_ \oplus u$$

$$f(f(a, e), u) = f(xa\_ , u)$$

As written, Involution removes the container of Object- $a$ , creating a typing violation for the first PUT argument since Object- $a$  may in fact contain several objects. To correct this, it is necessary PUT Object- $a$  into two empty containers. The need for the extra container is generated by the restriction on PUT arguments to be single objects.

$$\text{INVOLUTION} \quad (((a))) = (a)$$

$$(xu\_ (((xa\_)_a)_e)_e)_u = (xu\_ (xa\_)_a)_u$$

$$a \oplus e . \oplus e : \oplus u = a \oplus u$$

$$f(f(f(a, e), e), u) = f(a, u)$$

The Pattern-variable- $a$  in Pervasion also needs to be restricted to a single object. This restriction does not impair the implementation of Pervasion, since several objects can be operated on in sequence, or in the case of a parallel implementation, at the same time.

$$\text{PERVASION} \quad a (a \ b) = a (b)$$

$$(xu\_ a (a \ xv\_)_v)_u = (xu\_ a (xv\_)_v)_u$$

$$a \oplus v . \oplus . a \oplus u = v \oplus . a \oplus u$$

$$f(f(a, v), f(a, u)) = f(v, f(a, u))$$

### 8.6.3 Compositional Properties

The properties of commutativity, idempotency and zero can be identified across the composition of PUT operations when the function is augmented with LoF rules as constraints.

Temporal Commutativity:  $a \oplus . b \oplus u = b \oplus . a \oplus u$

$$f(a, f(b, u)) = f(b, f(a, u))$$

The independence of contained objects requires that there be no relation between those objects as they are PUT into a container. Therefore the order that objects are PUT into a container cannot matter.

Temporal idempotency is similar. Replication is a generalization of Calling.

REPLICATION  $a \ a = a$   $(xu\_ a \ a)_u = (xu\_ a)_u$

$$a \oplus . a \oplus u = a \oplus u \qquad f(a, f(a, u)) = f(a, u)$$

Like temporal commutativity above, Replication can be seen as a temporal, or compositional, idempotency. The form of idempotency is exhibited across the second application of PUT. PUT itself is not idempotent, because an object cannot be put into itself. It is possible, however, to PUT an object into a container and then PUT it in again. This composition of operations is idempotent. From the perspective of physical containment relations, the same object cannot be PUT into the same container twice. Thus, idempotency itself is a phenomenon of induced symbolism.

LoF also endows PUT with a compositional zero. The Position rule illustrates this zero,

$$a \oplus a \ . \oplus u = \emptyset \oplus u = u$$

That is, the composition  $a \oplus a$  is a void-equivalent left zero. It is also a right zero,

$$u \oplus . a \oplus a = u \oplus \emptyset = u$$

This "zero" rests on a notational deception. In the first place, we have defined new properties that permits an object to interact with nothing at all.

$$\begin{array}{ll} \emptyset \oplus u = u & \text{PUT nothing into Object-}u = \text{Object-}u \\ u \oplus \emptyset = u & \text{PUT Object-}u \text{ into nothing} = \text{nothing} \end{array}$$

These rules may sound appealing, but they do violate the fundamental premise of void-based iconic forms, that nothing does not participate in operations. Of course, that is exactly what the English translation of the symbolic forms says. But to reach these rules, the null token must be constructed and represented. And from the iconic perspective  $\emptyset$  does not have a location.

The deeper problem is that any void-equivalent form can serve as this type of zero. This zero is not unique, there are potentially an infinity of other forms of zero. Of course this is also true of the arithmetic of numbers, should compound expressions be permitted as elementary objects. For example, (1-1) is a compound zero. In the symbolic representation, the problem of compound zeros is addressed by constructing a token, "0" in this case, and the token itself is unique. However, since tokens are not native to iconic forms, the expedient of constructing one to stand in place of nothing is not available.

#### 8.6.4 LoF as PUT Operations

To better visualize the relationships between the functional LoF rules, Table 8.3 displays each rule (except Transposition which is not void-based, although it is one of many possible theorems), together with two notations for the functional form. The functional notation used throughout this section is one string notation; the other is a perhaps a more familiar algebraic notation.

In standard algebraic notation for magmas, an application of a function is represented by concatenation of the arguments, without writing the function itself. In the case of PUT,  $a \oplus b$  is written simply as  $ab$ . Until this section, we have recorded the alternative functional notation using the function variable  $f$ , as  $f(a,b)$ . The alternative notation also uses the null token  $\emptyset$  for visual convenience, and so that PUT applications can be minimized. In such cases, the rule is presented twice, in the longer and in the shorter versions. In general, variable naming follows these conventions,

$\emptyset$	null token
$e$	empty container
$a$	active pattern-variable
$u,v$	incidental structure

Finally, Quine dots are replaced by conventional grouping parentheses.

RULE	FUNCTIONAL FORM	ALTERNATIVE FUNCTIONAL NOTATION
INDEPENDENCE OF CONTENTS		
$(a\ b) = (b\ a)$	$a \oplus. b \oplus u = b \oplus. a \oplus u$	$a(bu) = b(au)$
CALLING		
$(\ )(\ ) = (\ )$	$e \oplus. e \oplus u = e \oplus u$	$e(eu) = eu$
REPLICATION		
$a\ a = a$	$a \oplus. a \oplus u = a \oplus u$	$a(au) = au$
DOMINION		
$a\ (\ ) = (\ )$	$a \oplus. e \oplus u = e \oplus u$	$a(eu) = eu$
CROSSING		
$((\ )) =$	$e \oplus e \cdot \oplus u = u$	$(ee)u = u$ $ee = \emptyset$
OCCLUSION		
$(a\ (\ )) =$	$e \oplus v \cdot \oplus u = u$	$(ev)u = u$ $ev = \emptyset$
POSITION		
$(a\ (a)) =$	$a \oplus a \cdot \oplus u = u$	$(aa)u = u$ $aa = \emptyset$
INVOLUTION		
$((a)) = a$	$a \oplus e \cdot \oplus e : \oplus u = a \oplus u$	$((ae)e)u = au$
PERVASION		
$a\ (a\ b) = a\ (b)$	$a \oplus v \cdot \oplus. a \oplus u = v \oplus. a \oplus u$	$(av)(au) = v(au)$

Table 8.3: LoF Rules as Properties of PUT

Structural examination of the alternative notation reveals these algebraic features of PUT constrained by LoF rules.

Symbolic inducement	Independence	$a(bu) = b(au)$
Non-associativity	Position Replication	$(aa)u = u$ $a(au) = au$
Void-equivalence	Crossing Occlusion Position	$ee = \emptyset$ $ev = \emptyset$ $aa = \emptyset$

Generalization	Calling	$e(eu) = eu$
	Dominion	$a(eu) = eu$
	Replication	$a(au) = au$
	Crossing	$(ee)u = u$
	Occlusion	$(ev)u = u$
	Position	$(aa)u = u$
	Crossing	$(\begin{smallmatrix} e & e \end{smallmatrix})u = u$
	Involution	$((ae)e)u = au$

The algebraic property equations in Table 8.3 are interdependent. For example, Pervasion can be stated in an alternative form via an application of the content independence property.

$$(av)(au) = a((av)u) = v(au)$$

As would be expected, the application of an induced property to a LoF rule results in inducing properties not originally in the LoF rule. The  $(av)(au)$  construction sequence PUTs Object-a into Object-v, and independently PUTs a replicate of Object-a into Object-u. Then the composite arrangement  $av$  is PUT into the composite arrangement  $au$ . This is the same as not putting Object-a into Object-v in the first place.

The  $a((av)u)$  construction sequence PUTs Object-a into Object-v, and then PUTs the composite  $av$  into Object-u. Then a replicate of Object-a is also PUT into the composite object. This too is the same as not putting Object-a into Object-v in the first place. The difference between the two representations is that one constructs in parallel and the other constructs them in sequence.

The LoF rule of Calling asserts that replicated objects are indistinguishable, the difference is illusory. Semantically, we cannot tell the difference between the original and the replicate, but due to the sequencing of the functional notation, the replicates are brought into being at different times. The string-based functional notation cannot help but make this distinction, so invariants must be asserted to remove the artificial distinction.

Another example of interdependence is that some of the axiomatic rules can be derived from others. For example, Position and Occlusion are equivalent in an axiomatic basis that includes Pervasion. Thus, there is considerable flexibility in selecting the algebraic properties that define PUT. The following selection is based on the LoF computational rules, although this set of properties is not necessarily anchored to other results in quasigroup theory.

$$\text{Independence of contents} \quad a(bu) = b(au)$$

Dominion	$a(eu) = eu$
Involution	$((ae)e)u = au$
Pervasion	$(av)(au) = v(au)$

Here is a proof of Replication based on Involution. The iconic proof is shown first, to provide a visual guideline. This proof is then transcribed into the alternative functional notation.

$a \quad a$	$= a$	to prove
$a \quad ( \quad a )$	$= a$	+Involution
$a \quad (a \quad (a))$	$= a$	+Pervasion
$a$	$= a$	-Position, identity
$a \quad (au)$		to reduce to $au$
$((ae)e) \quad (au)$		+Involution $au=((ae)e)u, \quad u/au$
$(a((ae)e))(au)$		+Pervasion $v(au)=(av)(au), \quad v/(ae)e$
$((ae)(ae))(au)$		Independence $a(bu)=b(au), \quad b/ae, \quad u/e$
$au$		-Position $(aa)u=u, \quad a/ae, \quad u/au$

The application of PUT functions to logic is presented in Section 10.3.

## 9 Remaining Modeling Issues

Several modeling issues remain; each emphasizes the differences between iconic and symbolic notations.

- Iconic notation incorporates the inherent parallelism available from spatial display, while symbolic notation incorporates the inherent sequencing of string-based display.
- The iconic use of space permits a natural display of nesting, while symbolic strings offload nesting to an iterative process. Iconic notations thus accommodate deep rules (rules that operate across nestings).
- While multiple reference to identical forms is restricted to identical labeling in symbolic notation, iconic form accommodates a wider use of structure sharing.
- Void-substitution provides new methods of computation that are not available in string-based representations.

- Isomorphic symbolic notations can be seen as relabelings; notational variants of iconic forms include topological and geometric transformation of the representation.

The void-based semantics of iconic representation differs significantly from the label-based semantics of symbolic representation. We intentionally use the word semantics to emphasize that, at least when broadening the idea of mathematical notation to include iconic forms, notation is never entirely separate from meaning. Even with purely symbolic forms, the notational substrate itself imposes restrictions on possible conceptualizations. Spencer Brown's widely misunderstood point is that the concepts of commutativity, associativity, and arity are not fundamental to mathematical thought, they are simply consequences of expressing mathematical ideas in a sequential rather than a parallel format. To paraphrase the central idea of linear logic, what is the cost of unrestrained symbol replication?

## 9.1 Inherent Parallelism

A representation is necessarily inscribed within some space. Textual representation is constrained to one-dimensional sequences. The spatial constraint of one dimension requires that symbols occur to the right or to the left of each other. Much of the analysis of mathematical expressions is conducted from this linear perspective. Group theory incorporates right- and left- relations. Complexity theory studies the consequences of string processing. The grammars of language theory are expressed as sequential assemblies of symbols. And the study of diagrammatic and spatial systems generally requires translation into string notation to achieve legitimacy.

Arrangements of containers incorporate both one- and two-dimensions of representation. All objects in the same container are structurally independent, they share a two-dimensional space bounded by their container. Nesting of containers is expressed across one dimension of depth. Therefore, arrangements support both parallel access and processing of contents, and sequential access and processing of nesting structures. Computational steps accumulate only when processing results are transferred across nested containers.

In the graph formalism, strong parallelism is achieved by enforcing no global communication between nodes. Each of the three computational initials of LoF can be enacted solely by local communication between adjacent nodes.

Here is an example of parallel and sequential steps together in the simplification of a parens arrangement,

[ (( ( ) ( ) ((( ))) )) ]  
(( )) Step 1: match Crossing



		(( ))	match Crossing
	( ) ( )		match Calling
[	( )	( )	apply in parallel
	( )	( )	Step 2: match Calling
[	( )		apply Calling

The first step finds three rule matches (two applications of Crossing and one of Calling). All three applications are applied at the same time to complete Step 1. Step 2 identifies one more application of Calling.

Symbolic notation was developed in an era prior to the concept of computational parallelism. The structure of the conventional mathematical language of relations is consequently biased toward the sequential mode of processing, even though representation is thought to be purely descriptive. Naturally, a parallel model requires a parallel processor to implement. In sequential machines, transformation is always implemented by a sequential, iterative process. This sequential perspective is reified in symbolic notation.

A set, conceptually at least, imposes no ordering or grouping principles on its members. The universe of valid pairs in any given relation presumes parallel access to every pair. It is the statement of relational constraints in the language of predicate calculus that imposes a sequential processing regime. The sequential model comes from a notational style of labeling all objects and then defining properties as structural constraints on locations of labels within the conjunction of relational pairs.

The parallelism of iconic forms makes pattern-variables an implementation technique rather than a necessary grouping mechanism. Only in sequential regimes do pattern-variables require an incidental grouping relation, such as membership in a set. That is to say, conventional grouping of arguments in relations is confounded with sequential processing over time. With parallel processing, the multiple bindings of pattern-variables are processed concurrently, independent of cardinality. In this case, structural grouping is superseded by structural independence. Thus, the necessity of an ancillary grouping relation within the description of containment depends upon the processing regime implicit within the notation used to express containment. The notation itself, at least when comparing symbolic and iconic languages, interacts with the concept of a computational step.

Spencer Brown is explicit about parallelism in LoF, although he does not use that word. Rather he identifies parallelism as “the freedom offered by an added dimension”. He contrasts spatial form to verbal form,

“.. in speech we can mark only the one dimension of time. Much that is unnecessary and obstructive in mathematics today appears to be vestigial of this limitation of the spoken word.”

Later, Spencer Brown discusses Sheffer's awkward stroke notation which expands the size of an expression exponentially rather than shrinking it as is fitting of a single operator notation.

"Sheffer explicitly assumes the restriction of his operator to binary scope, and also, implicitly, assumes the relevance of the order in which the variables under operation appear. Each of these assumptions is in fact less central to mathematics than is commonly supposed..."

Here then is Spencer Brown's remarkable idea made explicit. Group theoretic properties, that is, arity, commutativity and associativity, are incidental rather than fundamental. This perspective is available when the linear strictures of writing and speech are lifted, and mathematical relations (in particular, our model of containment) are expressed in iconic, spatial languages. Grouping is not at the core of the representational problem of converting LoF into predicate calculus. The deeper issue is that predicate calculus is formulated to be written with symbols on a line. The conversion of LoF into conventional notation is an exercise of undermining a parallel notation until it appears not as an image but as words. Table 9.1 lists the steps we have taken to convert the explicitly parallel notation in LoF into an explicitly sequential notation.

---

Convert spatial representation into symbolic representation via labeling.  
Label objects that represent themselves to provide a symbolic reference.  
Label nothing to represent negative space by a symbol.  
Construct symbolic links between objects that are not related spatially.  
Provide a grouping mechanism to assure no forms are independent.  
Package incidental structure into sets or with logical connectives.  
Construct operations for each different cardinality of the operands.  
Impose a sequential processing regime for consistency.

---

Table 9.1: Steps to Convert Parallel Notation into Sequential Notation

## 9.2 Deep Transformation Rules

Spencer Brown implicitly embraces parallelism within the contents of each container-object. However, he treats depth of nesting as purely sequential. For example, he extends Pervasion to a deep rule by these steps:

PERVASION                       $a (a \ b) = a (b)$                        $(x\_ y\_ (y\_ z\_ )_c )_u = (x\_ y\_ (z\_ )_c )_u$

$a ( \ b ( \ c (d \ \dots)))$   
 $a (a \ b ( \ c (d \ \dots)))$   
 $a (a \ b (a \ c (d \ \dots)))$   
 $a ( \ b (a \ c (d \ \dots)))$   
 $\dots$

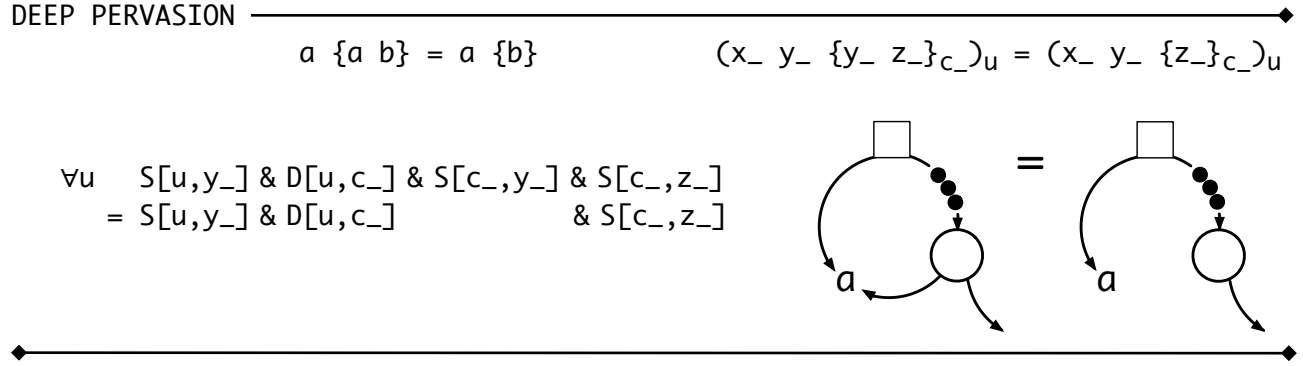
This sequence of steps permits Pervasion to be applied at any depth, so long as the pervading object is outermost.

A deep rule is one that applies across nesting, here across arbitrary container boundaries. In a sequential notation, deep rules ignore relational boundaries. They reach across the scope of relations to change both the arguments and the arity of other formulas.

Pervasion is the only deep rule in LoF. Pervasion is also of fundamental importance since it is the workhorse of LoF computation. It is possible to take this deep rule as fundamental to containment, rather than considering it to be an extension. From this perspective, Spencer Brown's shallow Pervasion (called Generation in LoF) is simply a special case. Deep rules are quite naturally expressed using the DeepContains relation. Such a perspective provides different insights into the LoF calculus.

### 9.2.1 Deep Pervasion

We will need to introduce a new notation. Let curly braces  $\{ \}$  represent any collection of deeper intervening container boundaries. This deep container is labeled with a pattern-variable, since it binds to any number of nested containers. The pattern-variable is a path-variable, standing in place of zero, one, or many container boundaries.



Like other pattern-variables in relational form,  $D[u, c\_]$  is a rule schema that articulates into specific rules for each different depth of nesting. Whereas relational descriptions are difficult to generalize to deep rules, graph descriptions are not. The only difference between the shallow and deep versions of Pervasion in graph notation is the additional vertical ellipsis that represents any arbitrary intervening graph structure.

### 9.2.2 Deep Replication

The rule of Replication expresses the idea that replicated forms are permitted (alternatively, can be deleted). Deep Pervasion generalizes Replication to any depth of nesting. The similarities between Replication and Pervasion can be displayed visually,

REPLICATION	$a \ a \ = \ a$	$(x\_ \ y\_ \ y\_ \ )_u = (x\_ \ y\_ \ )_u$
PERVASION	$a \ (a \ b) = a \ (b)$	$(x\_ \ y\_ (y\_ \ z\_ )_c )_u = (x\_ \ y\_ (z\_ )_c )_u$
DEEP PERVASION	$a \ \{a \ b\} = a \ \{b\}$	$(x\_ \ y\_ \{y\_ \ z\_ \}_{c\_})_u = (x\_ \ y\_ \{z\_ \}_{c\_})_u$

These three rules are each a special case of a single rule that incorporates a path-variable, that is, a variable that stands in place of zero, one, or many levels of nesting.

The function of parens and of curly braces is to keep objects that are not replicated in their place, that is, in the same containers. This interpretation aligns with the Physicality Constraint. The replication of labels permitted by Deep Pervasion does not construct overlapping containers, it simply embeds replicated forms deeper into a nested arrangement. That is, the nested depth of replication is not relevant to the process of replication. Due to the independence of objects as contents, the content of intervening containers is also not relevant. What is relevant is that the replicant is not placed outside of the context of the original.

It is the rule of Transposition that creates an illusion of overlap, by the replication of Variable-a in two separate containers.

$$\text{TRANSPPOSITION} \quad ((a \ b)(a \ c)) = a \ ((b)(c))$$

The rule itself shows how this overlap can be eliminated. However, Transposition is derivable from Pervasion (together with Involution and Dominion); this apparent violation of overlap can be constructed from non-overlapping nesting transformations. Transposition illustrates a tangle more than an overlap.

### 9.2.3 Permeability

Deep Pervasion suggests a permeability model of containers. Under the LoF rules, a container boundary is semi-permeable to objects outside that boundary, regardless of depth of nesting. That is, containers are transparent from the outside.

In the example below, Container-2 and Container-3 are both transparent with regard to Object-4. Deep Pervasion asserts that the presence of Object-4 at any deeper level of nesting is arbitrary. The inward facing boundaries can be treated as non-existent, a situation that is difficult to represent in both iconic and symbolic notations.

$$(4 \ (5 \ (4 \ 6)_3)_2)_1 \text{ as } (4 \ (5 \ (4 \ 6)_3)_2)_1 = (4 \ (5 \ (6)_3)_2)_1 \text{ as } (4 \ (5 \ (6)_3)_2)_1$$

Deep Pervasion might be expressed as

$$S[x,z] \ \& \ S[y,z] \ \& \ \text{Path}[x,y] = S[x,z]$$

The Path relation asserts successive nested containers between Container-x and Container-y. It takes the form of successive labels alternating between range and domain:

$$\text{Path}[x,y] \text{ =def= } (((\dots((z)_y)_k \dots)_b)_a)_x = \{(x,a),(a,b),\dots,(k,y),(y,z)\}$$

The Path relation, however, is equivalent to a step-wise traversal of a nested arrangement, and does not achieve the intention of the concept of transparency. What is needed, instead, is something like of the transitive closure of DeepContains. In a graph, transitive closure adds additional links so that every possible transitive path is converted into a single direct path. Analogously, Pervasive closure would add links that represent all possible deep nestings of replicates. From the graph viewpoint, closure converts paths to direct links. From the symbolic viewpoint, closure identifies the complete set

of valid pairs within a transitive relation. Finally, from the perspective of semipermeable boundaries (i.e. Deep Pervasion), both paths and transitive relational formulas are notational artifacts. Transparent objects are not discernible, so that they do not support identification and labeling.

Deep Pervasion identifies temporary rather than permanent void-equivalent structure. Operationally, void-equivalent containers are deleted temporarily during transformation and then reconstructed after transformation.

### 9.3 Structure Sharing

Structure sharing is the process of using the same label to identify the same compound expression in different locations within a larger expression. In any pattern-matching implementation, or any implementation that uses symbolic labels, structure sharing is an essential aspect of identifying identical patterns that may have different labels. Condensing a structure by labeling identical structures with the same label is also necessary for addressing large problems, for instance, optimizing the routing of million gate silicon circuits.

Structure sharing in symbolic notations usually occurs through explicit labeling of equivalent objects. Labeling is strictly controlled by assigning different labels to each different structure. Whenever two identical structures are identified, they can be assigned the same label. This label usually participates in transformation as a discrete object, with the implicit constraint that the transformation is restricted to operate on the label itself, and not on the compound form identified by the label. In almost all descriptions of symbolic transformation, structure sharing is not mentioned, and is relegated to an implementation detail.

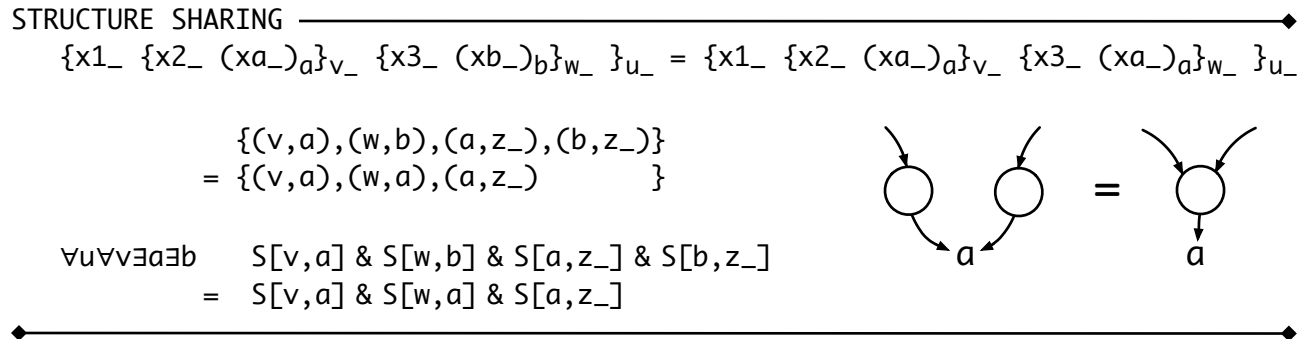
Structure sharing is necessary in iconic notations to identify compound arrangements that participate in transformation rules. For example,

$$(a \ b \ c) \ (a \ b \ c) = (a \ b \ c) \quad \text{Replication}$$

Replication allows deletion of one of the two copies of the arrangement  $(a \ b \ c)$ . Visually it is easy to see that the two arrangements are identical, however an implementation may not have identical labels for these arrangements. The data structure representing this reduction may be  $(a \ b \ c)_1 \ (a \ b \ c)_2$ . In this case, it is not apparent that the two arrangements are identical. Structure sharing assigns the same label or pointer to containers with the same contents.

#### 9.3.1 Structure Sharing Transformation

Structure sharing is described below in annotated parens, graph, and symbolic notations. The parens version is not listed because parens representation does not include labeled boundaries to support structure sharing.



Above, two objects/nodes contain the same contents, Object-a. The two nodes are occur as different parts of an arrangement. In the graph notation, references to the separate but identical sub-arrangements are combined into multiple references to a single object.

The annotated parens notation for structure sharing is particularly awkward, since structure sharing essentially addresses mislabeling. Symbolic identification of mislabeling (i.e. the misuse of symbolic labels) is tricky to state formally in symbols. The three sets of curly braces express the idea that structure sharing can occur between identical sub-arrangements anywhere with a larger arrangement, regardless of depth of nesting (the path-variables  $u\_ , v\_ ,$  and  $w\_$ ) or the presence of other contents (the pattern-variables  $x1\_ , x2\_ ,$  and  $x3\_$ ). The contents of Object-a and Object-b (the pattern-variable  $z\_$ ) are identified so that the two objects can be determined to be identical, but under different labels.

In symbolic notation, structure sharing is usually accomplished in the meta-language, aside from the symbol structure of interest. New labels are introduced and then substituted, as in this example,

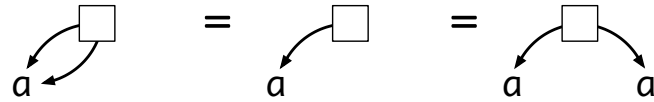
$(x\_ (a\ b\ c)\ (y\_ (a\ b\ c)))$

Let  $z = (a\ b\ c)$ ; substituting  $z$  for  $(a\ b\ c)$  yields

$(x\_ z\ (y\_ z))$

Although structure sharing is applied to a single arrangement, in symbolic notations it necessarily incorporates additional equations in the form of label identities. The identification of identical structures in a symbolic string is often done by inspection. In this case the symbolic string is treated iconically, and the visual identification is an iconic transformation.

Structure sharing and replication are intimately connected, as is illustrated using graph notation,



The middle figure shows a single arrangement labeled *a*. It can be replicated in two distinct ways. On the left, the replication is of a link to Object-*a*; on the right the structure of Object-*a* itself is replicated. Structure sharing is a bottom-up process of replacing identical objects with multiple links, similar to the transformation between the left and the right figures above. Unlike replication, identical objects are combined rather than deleted.

### 9.3.2 Inconsistency of Reference

Since LoF rules are algebraic, they are reversible. This means that it is also possible that, at some point, structures with the same label can become differentiated, requiring different labels. This can occur when the same labeled arrangement is placed into two different containers. Transposition is an example:

$$\text{TRANSPPOSITION} \quad ((a \ b)(a \ c)) = a \ ((b)(c))$$

In condensing the reference to Variable-*a* from two occurrences to one (reading left to right), consistency of reference is assured. However, splitting reference to Variable-*a* from one to two occurrences (reading right to left) can result in two arrangements with identical labels that no longer have an identical structure. For example, this application of Transposition replicates Container-1.

$$\begin{aligned} [(5 \ 6)_1 \ ((5)_3(6)_4)_2] &= [((5 \ (5 \ 6)_1)_3 \ (6 \ (5 \ 6)_1)_4)_2] && \text{transposition} \\ &= [((5 \ ( \ 6)_1)_3 \ (6 \ (5 \ )_1)_4)_2] && \text{pervasion} \end{aligned}$$

Local application of Pervasion then changes the contents of each Container-1, so that the replicated containers now have different contents. The result is an inconsistency of reference.

The relational description of the above example also illustrates this problem:

$$\begin{aligned} \{(u,1),(u,2),(1,5),(1,6),(2,3),(2,4),(3,5),(4,6)\} &\implies \\ \{ \quad (u,2), \quad \quad \quad (2,3),(2,4),(3,5),(4,6), \end{aligned}$$



$$(3,1),(1,5),(1,6), \\ (4,1),(1,5),(1,6)\} \Rightarrow$$

In the first step, Transposition constructs two replicates of Container-1. The idempotency of conjunction suggests that the replicated pairs (1,5) and (1,6) are redundant. Applying Pervasion as the next step however, creates a notational error in the relational structure,

$$\{ \quad (u,2), \quad (2,3),(2,4),(3,5),(4,6), \\ (3,1), \quad (1,6), \\ (4,1),(1,5)\}$$

since this reads that Container-1 still contains both Object-5 and Object-6. That is to say, the context of each of the replications is lost. The above relational structure still describes the arrangement,

$$[ \ ( \ (5 \ (5 \ 6)_1 \ )_3 \ (6 \ (5 \ 6)_1 \ )_4 \ )_2 \ ]$$

This reading is necessary because we have elected to interpret the pairs (1,5) and (1,6) as  $(5 \ 6)_1$ . How then should replication be represented? A direct solution is to relabel replicated containers whenever a change in contents is effected. Another option is to relabel containers whenever they are replicated, however this is a rather extreme choice since often the purpose of replication is to indicate the same arrangement.

The thematic issue is whether the rules for relabeling are an essential aspect of mathematical description, or simply a notational artifact. Visual comparison of arrangements suffices for small iconic structures, however all notations that rely on labeling, whether iconic or symbolic, must address the inverse issues of identical structure under different labels, and identical labels for different structures.

## 9.4 Void-based Computation

Void-equivalent arrangements can be freely constructed and deleted during computation. However, since these forms are equivalent to nothing at all, it is not necessary to explicitly replicate the iconic representation during construction. Rather, a void-equivalent form can be considered to be virtual.

A virtual arrangement is a void-equivalent form that is postulated to be present in order to effect reduction. Should the arrangement result in a desired reduction, then the virtual arrangement is considered to be present. Should a desired reduction not occur, then the virtual arrangement is simply treated as never having existed. Another way to phrase this is that a virtual arrangement

asks a structural question, without commitment to replication or to increasing the amount of explicit notation. Should the question be answered in the positive, then the virtual arrangement is used. Should the question be answered in the negative, then nothing further occurs, the virtual arrangement has not been replicated, and there is no change of structure.

This technique has no equivalent in symbolic computation. Since symbols are always taken to be explicit, symbolic structures can be modified only explicitly, and the modified structure that remains cannot arbitrarily be deleted. There is also no conventional mechanism to incorporate query structure within the representation of relations, since such queries are seen to be outside the relational description, as part of the proof technique itself. (An exception is query variables in pattern-matching regimes.) Virtual structure is another example of the blurring of syntax and semantics within iconic notations.

Void-based technique using virtual arrangements is illustrated below in a proof of the Resolution theorem of propositional logic. The Pervasion rule asserts that replicates can be constructed in any nested space. These replicates can be virtual; virtual arrangements are contained within carets in the notation. A plus token, +, in front of a rule indicates that it has been applied in the constructive direction; a minus token, -, indicates that the rule has been applied to delete structure.

The example shows two different applications of virtual Pervasion, one unsuccessful, one successful. There are few structural clues that would indicate one attempted application should be preferred over the other. In the unsuccessful application, the virtual arrangement, after undergoing modification, is simply ignored as not useful.

RESOLUTION	$((a\ b)\ ((a)\ c)) = ((a\ b)\ ((a)\ c)\ (b\ c))$	
	$((a\ b\ )\ ((a)\ c)\ (b\ c))$	rhs
	$((a\ b\ ^((a)\ c)\ (b\ c)^)\ ((a)\ c)\ (b\ c))$	+virtual Pervasion
	$((a\ b\ ^(( )\ c)\ ( c)^)\ ((a)\ c)\ (b\ c))$	-Pervasion a b
	$((a\ b\ ^\ ( c)^)\ ((a)\ c)\ (b\ c))$	-Occlusion
	$((a\ b\ )\ ((a)\ c)\ (b\ c))$	failure, ignore virtual

In the successful application, the virtual arrangement, after undergoing modification, triggers a reduction that proves the desired equivalence.

$((a\ b)\ ((a)\ c)\ ( \quad \quad \quad b\ c))$	rhs
$((a\ b)\ ((a)\ c)\ (^{(a\ b)}\ ((a)\ c)^{b\ c}))$	+virtual Pervasion
$((a\ b)\ ((a)\ c)\ (^{(a\ )}\ ((a)\ )^{b\ c}))$	-Pervasion b c
$((a\ b)\ ((a)\ c)\ (^{(a\ )}\ ( \quad \quad )^{b\ c}))$	-Pervasion (a)
$((a\ b)\ ((a)\ c)\ ( \quad \quad \quad ))$	-Occlusion

Void-based techniques are particularly useful when applied to minimization of arrangements, that is, to reconfiguring arrangements so that they contain the minimal structure. Under an interpretation for logic, this would be called Boolean minimization. For example, the proof of equivalence above may have been stated in a different format, "Minimize the structure  $((a\ b)((a)\ c)(b\ c))$ ". In minimization, the specific goal is not known, while in proof, the goal is known. This makes minimization much more difficult than proof.

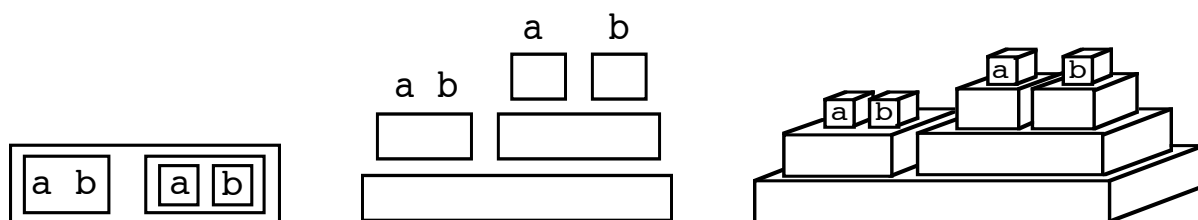
## 9.5 Iconic Languages

Thus far, we have shown two iconic languages, parens and graphs. Iconic languages can be transformed into other visual forms. However this translation occurs by geometric transformation, not by relabeling. The multiple interpretations of a symbolic system become multiple notations within an iconic system. Although the various iconic languages would map onto the same relational structures, the different spatial displays of these languages would conventionally be identified with different interpretations of that relational structure.

We present two additional iconic notations for the LoF calculus. Each involves a geometric transformation of containment arrangements. The transformation rules of the LoF iconic system remain unchanged.

### 9.5.1 Stacked Blocks

Parens are truncated two-dimensional containers. The steps to convert containers to stacks of three-dimensional blocks consist of spatial extrusions.



For the parens form  $((a\ b)((a)\ b))$ , for example, parens are first capped to form boxes, i.e. two-dimensional enclosures. These are rotated out of the plane of the page by  $90^\circ$ . Or one could say that the box form is extruded upwards. Boxes are made solid by extruding the stacked two dimensional form into a third dimension perpendicular to the plane of the page. The resulting solid blocks constitute a formal system that supports computation by direct and visceral manipulation of physical representations.

Figure 9.1 shows the representation of Crossing and Calling in the iconic block notation.

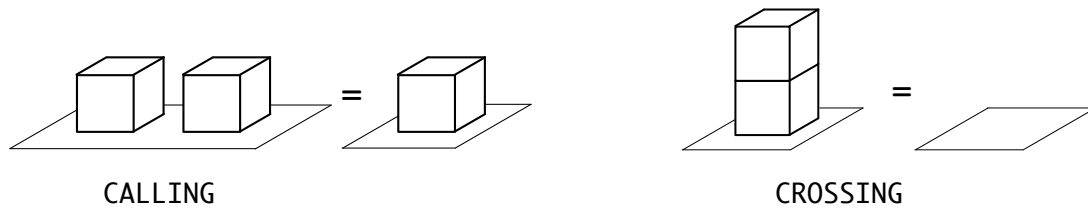


Figure 9.1: Arithmetic Initials Expressed as Stacked Blocks

The universal container is now the table upon which the stacked blocks rest. More generally, the supporting plane stands in place of any lower block. The null token is an explicitly empty top of a block. The relational description remains the same, however the interpretation of the relation changes from Container-a Contains Container-b to Block-a Supports Block-b.

Figure 9.2 shows Spencer Brown's initials for the LoF algebra in block notation. Pattern-variables are represented as inverted cones to emphasize that they stand in place of any configuration of blocks, including the absence of blocks and multiple stacks of blocks. The inverted cone means "anything can be placed here". When different collections of those anythings are being partitioned for transformation, the inverted cones are labeled.

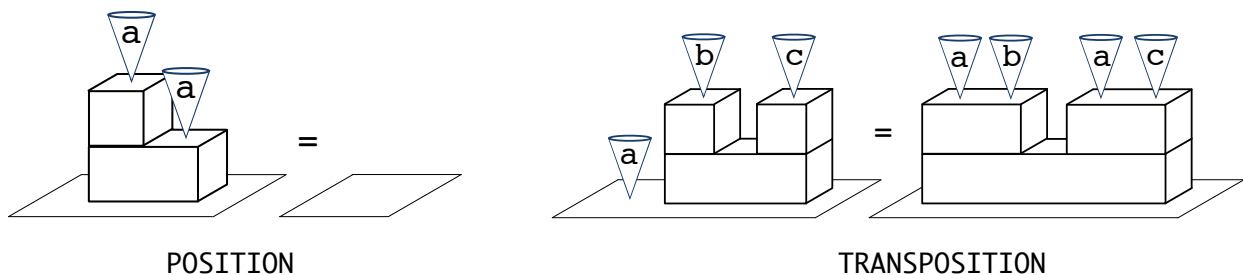


Figure 9.2: Algebra Initials Expressed as Stacked Blocks

The computational initials for the LoF algebra are presented in Figure 9.3.

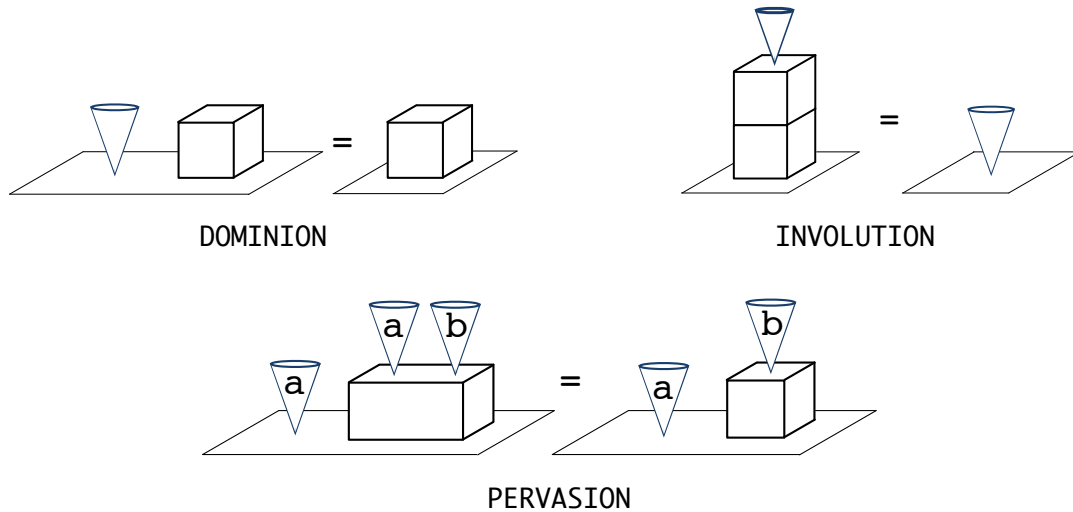
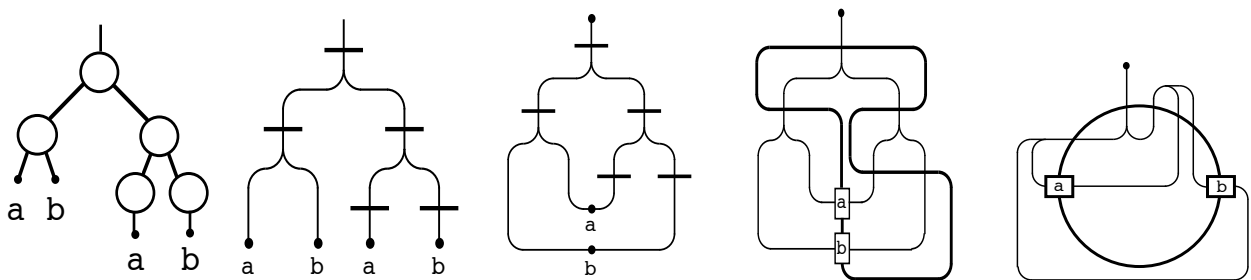


Figure 9.3: Computational Initials Expressed as Stacked Blocks

### 9.5.2 Paths

As another example of iconic notation, Kauffman identifies a simple transformation that converts any arrangement of containers into a process of entering and leaving a single container [ref]. An informal illustration of the conversion from trees to paths follows.

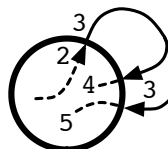


The first figure, a tree, is the parens form  $((a\ b)((a)(b)))$  extruded downward (Section 3.4 contains details). Nodes are converted into bars, and the downward links of multiple children nodes are merged into diverging paths. The tree is then converted into a graph via structure sharing of labels. A path is drawn connecting all bars and labels in a closed loop. A formal procedure exists for this transformation. The final figure is cosmetic, the closed loop is shaped into a circle.

The path representation explicitly converts nodes as objects into crossings as actions. Traversal of the paths is equivalent to traversal of the graph; it is

easy to see that the link structure of the graph is converted without modification into the path structure. Each path segment, between crossings, is identified by a container label in the ordered pairs form. Crossing the boundary changes the label.

$\{ \dots (2,3), (3,4), (3,5) \dots \}$ :



Names then are no longer associated with nodes, they are associated with path segments between crossings. The name associated with the parent node becomes the name of the path exiting downward from the parent node. A path segment depicts all references to the particular label, references that are dispersed throughout the relational formulas that would describe an arrangement. Path segments unify replicated symbolic labels into a single representation. Splitting or diverging paths illustrate label replication within a symbolic form. Paths themselves trace nestings rather than relations between objects.

From the perspective of ordered pairs, a path segment exiting the single boundary leaves as a member of the range. When it meets the boundary again, it meets as a member of the domain of the relation. In this way, a label participating within both the range and domain is integrated into a contiguous object. An appropriate name for the new iconic relation is CrossesInto: Path-1 CrossesInto Path-2.

A weakness of the path notation is that labels must rest on the border, they indicate the input nodes of a graph. The border itself `[[FINISH]]`

### 9.5.2.1 Iconic Rules for Path Notation

Figure 9.4 shows the two arithmetic initials of LoF in path notation. Directional arrows are attached to paths for ease of reading. The universal container is represented by a small square, on the outside of the boundary. More generally, the square represents any position along any path prior to the path-pattern identified as emerging from the square. A path meeting the boundary but not crossing it represents the empty container. A null token is not necessary. Nesting is represented by a path that crosses the heavy boundary. Multiple contents are represented by diverging paths.

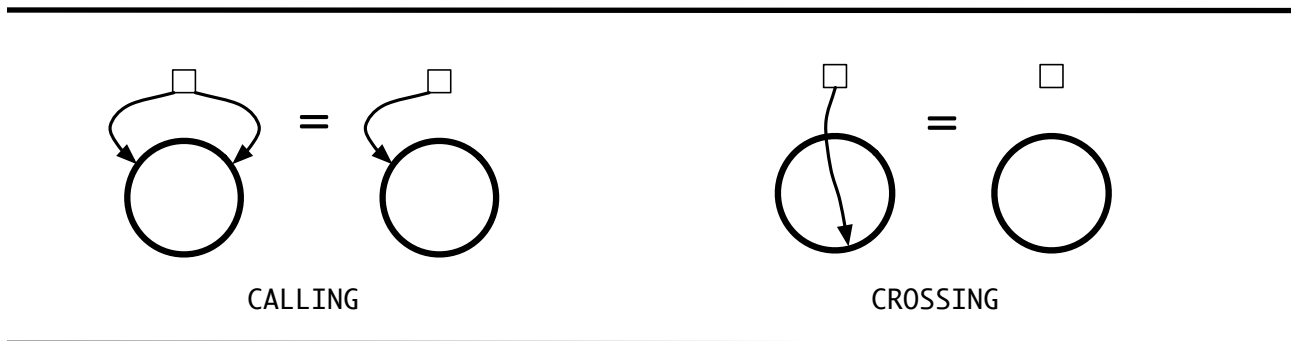


Figure 9.4: Arithmetic Initials Expressed as Paths

Figure 9.5 shows the path representation of Spencer Brown's two initials for the algebra. Small letters represent pattern-variables, standing in place of any path continuation, including path termination (no contents) and multiple path continuations shown as path splitting (multiple contents).

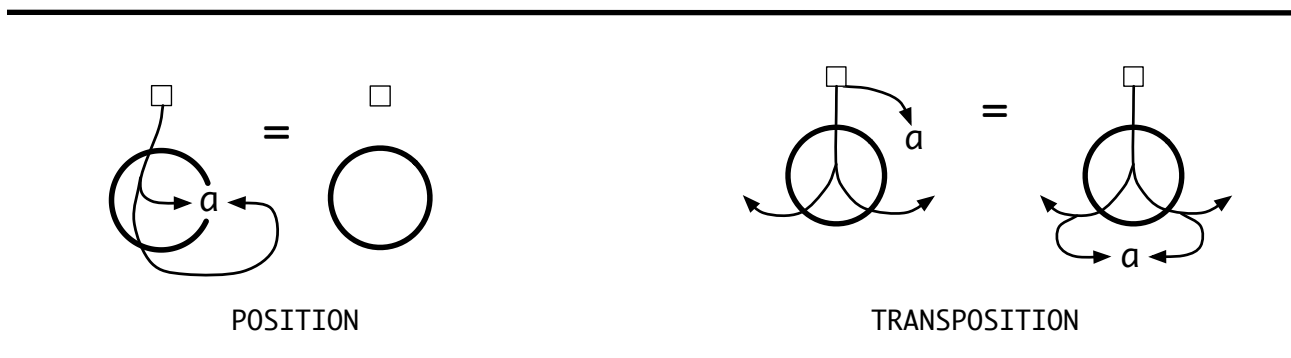


Figure 9.5: Algebra Initials Expressed as Paths

In Position, the variable label on the boundary represents any path continuation. The continuation must be replicated on each side of the boundary. The iconic structure of Transposition is sufficient to differentiate pattern-variables without labeling. The interpretation of this iconic notation is invariant under geometrical transformation such as translation, rotation and mirroring.

Figure 9.6 shows the computational initials of the LoF algebra. Again, path labels are not strictly necessary.

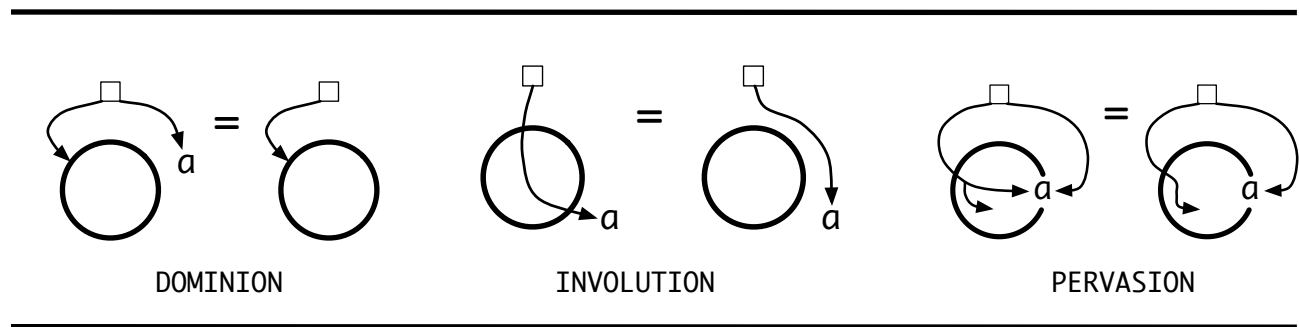


Figure 9.6: Computational Initials Expressed as Paths

### 9.5.2.2 Perspective

There is no preferential side of the boundary, path segments exist on both the "outside" and the "inside". This characteristic emphasizes an aspect of iconic containment not yet mentioned. The curvature of the boundary indicates the location of the observer, or reader, of the boundary. Conventionally the universal container and the location of the reader are co-defined. However, there is no structural reason that one side of a boundary be treated preferentially. The directionality of the graph representation, for example, can be from top to bottom (placing the reader on the outside), or from bottom to top (placing the reader on the inside). Similarly, what is called the context is on the same side of the boundary as is the reader. What is called the content is on the other side. This situation is analogous to the visual choice of perspective provided by a Necker cube.

Iconic notation thereby incorporates a choice of observational perspective, in contrast to symbolic notation which does not. However, symbolic notation can easily be reoriented. In the containment model, the Contains relation would change into an IsContainedBy relation, and  $U$  and  $\emptyset$  would exchange roles. That is, symbolic notation handles perspective by constructing inverse relations. Iconic notation simply changes the perspective of the reader. The choice of reading perspective is similar to the figure-ground reversal available in most images.

## 10 Logic and Boolean Algebra

The simplicity of pattern-matching iconic forms suggests that some fundamental mathematical concepts might be more conveniently conceptualized in an iconic notation that itself resembles those mathematical concepts. If the interpretation of LoF were to be constrained to something as overtly visual as containers or graphs, the idea of iconic representation may possibly be non-



controversial. However, Spencer Brown includes a one-to-many map from iconic enclosures to propositional logic. We are thus confronted with the interesting idea that logic itself has an elegant iconic foundation, a formal system that resembles what it represents. Logic, that bastion of symbolic thought, has an effective visual model. This suggests that our current ideas about rationality might require reconsideration. Since LoF provides a new method of deduction, Spencer Brown has indeed created something controversial.

Our detailed examination of iconic and symbolic form rests upon this deeper motivation, the interpretation of LoF as logic. Logic has been the focus of intellectual interest for millennia. The logical connectives are deeply embedded in human languages and speech, and have only recently been associated with symbolic forms (in G. Boole's *The Laws of Thought*, 1854 [ref]).

C.S. Peirce first identified the iconic structure of logic with his Existential Graphs (c. 1880) [ref]. His work parallels Spencer Brown's, with the exception that LoF formulates iconic logic in an equational format while Existential Graphs are formulated in an implicative format. These logicians considered the iconic representation of logic to be both natural and necessary.

LoF is a pictorial logic that is expressively equivalent to symbolic logic, while at the same time being both computationally more elegant and experientially more accessible. It is known that symbolic logic is very difficult for students, and at times counter-intuitive. What has not yet been explored is the possibility and consequences of thinking both rationally and formally not with words but with images. Again, we return to the theme that meaning is not separate from notation, that how we think is not separate from the tools that we use to think with.

Spencer Brown shows that the transformation of iconic containment arrangements can be mapped to propositional logic. Therefore, the pattern-matching mechanism that implements LoF rules also implements logical deduction. Table 10.1 shows the many-to-one map from propositional logic to parens, and the one-to-one map from logic to annotated parens and to ordered pairs.

LOGIC	PARENS	ANNOTATED PARENS	ORDERED PAIRS
FALSE		$[\emptyset]$	$\{(U, \emptyset)\}$
$a$	$a$	$[a]$	$\{(U, a)\}$
TRUE	$( )$	$[(\emptyset)_c]$	$\{(U, c), (c, \emptyset)\}$
NOT $a$	$(a)$	$[(a)_c]$	$\{(U, c), (c, a)\}$

$a \text{ OR } b$	$a \ b$	$[a \ b]$	$\{(U,a),(U,b)\}$
$a \text{ IMPLIES } b$	$(a) \ b$	$[(a)_c \ b]$	$\{(U,c),(U,b),(c,a)\}$
$a \text{ AND } b$	$((a)(b))$	$[((a)_d(b)_e)_c]$	$\{(U,c),(c,d),(c,e),(d,a),(e,b)\}$

Table 10.1: The Map from Logic to Parens and to Ordered Pairs

This map provides yet another interpretation of containment, as logical implication. The arrangement  $(2)_1$ , reads

1 Contains 2  
1 Supports 2  
1 CrossesInto 2  
1 IstheParentOf 2.

It also reads: 2 IMPLIES the-context-of-1.

## 10.1 Deconstructing Logic

The interpretation of containment as implication necessarily involves two containers. In the simplest case, this can be recorded as

IMPLIES:  $[x_ \ (2)_1]$   $2 \rightarrow x_$

In the case of  $[(2)_1]$ ,  $x_$  is bound to nothing. This form might be interpreted as 2 IMPLIES  $\emptyset$ . More conventionally, it would be interpreted as NOT 2. In the case of  $[3 \ (2)_1]$ ,  $x_$  is bound to one object, and is interpreted as 2 IMPLIES 3. In the case of  $[3 \ 4 \ 5 \ (2)_1]$ ,  $x_$  is bound to several objects; 2 IMPLIES 3 OR 4 OR 5. The container of any arrangement can be interpreted as implication, so that implication can be read from multiple perspectives. For example,  $[(4)_3 \ (2)_1]$  can be interpreted for logic as 2 IMPLIES 3, and as 4 IMPLIES 1.

Within the iconic form, there is no preferential structure of implication. This is quite different than the perspective of implicative logic, which builds outward from antecedents to conclusions. Pervasion makes this contrast quite clear, it is the outer arrangements that dominate their inner replications. From the perspective of Pervasion, deduction proceeds not from antecedent to consequent, but from outer arrangement to inner replication. Iconically, conclusions dominate premises. A substantive comparison example is provided in Section A.7.

The sixteen binary Boolean connectives are interrelated as the vertices of a hypercube lattice. However, the symbolic form of the Boolean connectives is opaque to this structure. When converted into containment arrangements, the structural relationships are more apparent. The Boolean hypercube with vertices expressed in iconic notation is shown in Figure 10.1. With the exception of Boolean equivalence, the structure of the Boolean connectives matches the ways that two objects can be put into containers, separately and together. The form of Boolean equality nests the two objects at odd and even depths. There are two ways to nest two objects, either nested form becomes the other when put into another container.

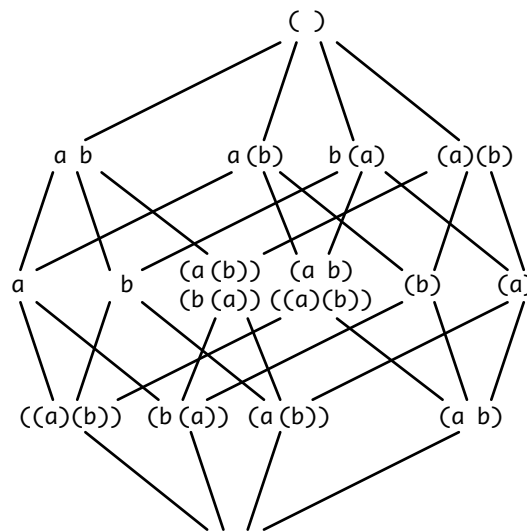


Figure 10.1: The Boolean Hypercube with Parens Vertices

## 10.2 Interpretation as NOR

There are two binary connectives that provide a single operator basis for Boolean logic, the Sheffer stroke and the Peirce/Quine dagger. In Computer Science, these would be called NAND and NOR respectively. We consider here the interpretation of a container as a generalized NOR operator that accommodates any number of arguments.

NOR is functionally complete, all other binary Boolean operators can be expressed in terms of NOR alone. Binary NOR, represented by the Peirce dagger  $\dagger$ , is able to express single argument Boolean operators, such as NOT, using replication of labels. For example,  $\neg p$  transcribes into  $p\dagger p$  (i.e.  $p$  NOR  $p$ ). It takes the addition of an assumed tautology, such as  $p \rightarrow p$ , or  $p \vee \neg p$ , or even  $p \equiv p$ , to express constants using dagger notation exclusively.

Dagger FALSE:

$p\dagger p:\dagger:p\dagger p.\dagger.p\dagger p$

$p \vee \neg p$

Dagger TRUE:  $p \dagger p . \dagger . p : \dagger : p \dagger p . \dagger . p$   $p \rightarrow p$

Five connectives, six replications of an irrelevant label, and an assumed tautology are required to identify the constant TRUE using binary NOR connectives. Certainly, the Spencer Brown distinction does not map one-to-one onto the Peirce dagger.

Table 10.2 shows a broadened definition of NOR as a variary operator, and its correspondence to parens and to dagger notations.

PARENS	NOR	DAGGER
( )	NOR[FALSE] = TRUE	$a \dagger a . \dagger . a : \dagger : a \dagger a . \dagger . a$
(a)	NOR[a] = NOT[a]	$a \dagger a$
(a b)	NOR[a,b]	$a \dagger b$
(a b c ...)	NOR[a,b,c,...]	$a \dagger b . \dagger . c : \dagger : \dots$

Table 10.2: Generalized NOR as an Interpretation of Containment

The correspondence between parens and the generalized NOR is appealing, but not at all isomorphic. In particular, NOR can be interpreted as NOT OR, the negation of a disjunction. The containment interpretation of distinction does not include disjunction. Objects inside a container are independent rather than joined by a truth theoretic operator.

The generalization of NOR itself to be a variary operator is not compatible with conventional group theoretic systems, since a hallmark of relational theory is that relations have a specific fixed arity. The non-correspondence of LoF and generalized NOR logic becomes apparent when the arithmetic of LoF is transcribed into generalized NOR and then retranscribed back into parens, as illustrated in Table 10.3.

PARENS	GENERALIZED NOR	RETRANSCRIBED PARENS
( ) ( ) = ( )	$\text{NOR}[\text{NOR}[\text{NOR}[], \text{NOR}[]]] = \text{NOR}[]$	$(( ( ) ) ( ) ) = ( )$
(( )) =	$\text{NOR}[\text{NOR}[]] = \text{NOR}[\text{NOR}[]]$	$(( ( ) ) = (( ( ) )$

Table 10.3: LoF Arithmetic as NOR Logic

In Calling, there is no direct way in NOR logic to express what might be interpreted as the disjunction of two containers, each of which stand in place of the constant TRUE. In Crossing there is no direct way to express void. The retranscriptions show that NOR logic cannot directly express parens logic. The circumstance is quite similar to parens arrangements requiring a universal container in order to be expressed symbolically, and in fact, NOR logic augmented by U does not exhibit the retranscription problem. It does, though, exhibit the arity problem.

By incorporating the substrate (i.e. void) into the notation, LoF becomes both more compact and more efficient than a single operator Boolean logic. However, LoF challenges the conventional concept of expressiveness. LoF has no representation of the concept FALSE, so technically it is less expressive than Boolean logic. FALSE is represented indirectly as NOT TRUE, (( )).

### 10.3 Logic as PUT Functions

The sixteen Boolean functions that define logical operations can be expressed both as containment relations and as compositions of PUT functions. Since the logical connectives can each be expressed using one binary function, the minimal basis for logic is this single function,  $\langle A, \oplus \rangle$ , applied to the set of parens arrangements, A.

#### 10.3.1 The Logical Quasigroup

Table 10.4 shows the sixteen binary Boolean functions, expressed as compositions of PUT functions.

BOOLEAN	PARENS	PUT FUNCTIONS	ALTERNATIVE NOTATION
$\emptyset$		$e \oplus e = U$	$ee = \emptyset$
NOT a	(a)	$a \oplus e$	$ae$
NOT b	(b)	$b \oplus e$	$be$
a NOR b	( a b )	$b \oplus . a \oplus e$	$b(ae)$
a NIF b	( a (b))	$b \oplus e . \oplus . a \oplus e$	$(be)(ae)$
b NIF a	((a) b )	$a \oplus e . \oplus . b \oplus e$	$(ae)(be)$
a AND b	((a)(b))	$a \oplus e : \oplus : b \oplus e . \oplus e$	$(ae)((be)e)$
$a \neq b$	((a)(b))(a b))	$a \oplus e : \oplus : b \oplus e . \oplus e : \oplus : b \oplus . a \oplus e : \oplus e$	$((ae)((be)e))(((b(ae))e)$
$a = b$	((a)(b))(a b)	$a \oplus e : \oplus : b \oplus e . \oplus e : \oplus : b \oplus . a \oplus e : \oplus U$	$((ae)((be)e))(((b(ae))U)$

$a \text{ NAND } b$	$(a)(b)$	$a \oplus e : \oplus : b \oplus e . \oplus U$	$(ae)((be)U)$
$b \text{ IF } a$	$(a) b$	$a \oplus e . \oplus . b \oplus U$	$(ae)(bU)$
$a \text{ IF } b$	$a (b)$	$b \oplus e . \oplus . a \oplus U$	$(be)(aU)$
$a \text{ OR } b$	$a b$	$b \oplus . a \oplus U$	$b(aU)$
$b$	$b$	$b \oplus U$	$bU = b$
$a$	$a$	$a \oplus U$	$aU = a$
$1$	$( \quad )$	$e \oplus U$	$eU = e$

Table 10.4: Binary Boolean Operators as PUT Functions

Figure 10.2 shows the Boolean hypercube with vertices labeled as compositions of PUT functions.

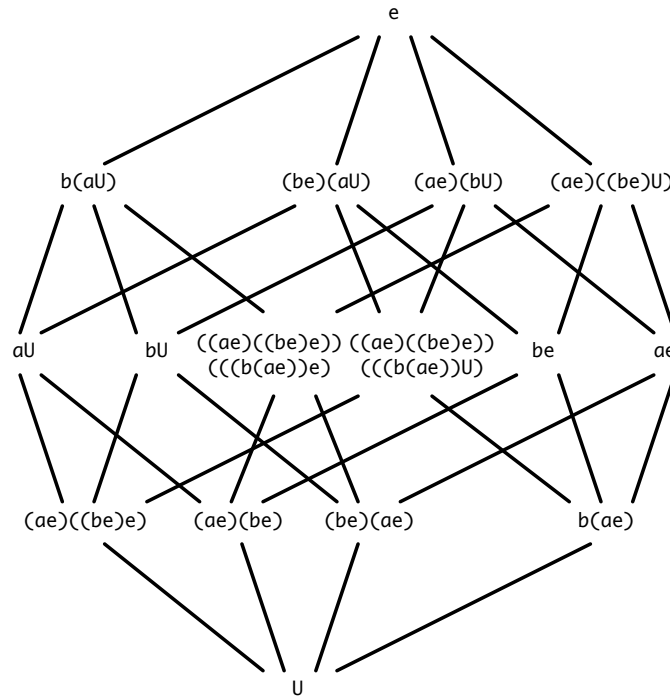


Figure 10.2: The Boolean Hypercube Labeled with PUT Compositions

This lattice shows that the complement of any PUT composition is formed by replacing  $U$  by  $e$  as the outer container. However, since the vertices form a quasigroup, the expected lattice structure of meets and joins is not in evidence. This hybrid figure then is deceptive, since it confounds two completely different models of Boolean algebra.

### 10.3.2 Quasigroup Deduction

It is now a simple transcription process to convert the well-known Boolean identities into their corresponding algebraic quasigroup expressions. Table 10.5 shows several of these symbolic Boolean identities, together with their iconic parens forms and their functional PUT forms.

BOOLEAN	PARENS	PUT ALTERNATIVE NOTATION
$p \& p = p$	$((p)(p)) = p$	$(pe)((pe)e) = p$
$p \vee p = p$	$p \ p = p$	$p(pU) = p$
$p \& q = q \& p$	$((p)(q)) = ((q)(p))$	$(qe)((pe)e) = (pe)((qe)e)$
$p \vee q = q \vee p$	$p \ q = q \ p$	$p(qU) = q(pU)$
$\neg\neg p = p$	$((p)) = p$	$(pe)e = p$
$p \& . p \vee q = p$	$((p)(p \ q)) = p$	$(pe)((p(qe))e) = p$
$p \vee . p \& q = p$	$p \ ((p)(q)) = p$	$(qe)((pe)e)(pU) = p$
$\neg p \& p = \neg q \& q$	$((((p)))(p)) = (((q))(q))$	$((qe)e)((qe)e) = ((qe)e)((qe)e)$
$\neg p \vee p = \neg q \vee q$	$(p) \ p = (q) \ q$	$(pe)(pU) = (qe)(qU)$
$\neg . p \vee q = \neg p \& \neg q$	$(p \ q) = (((p))((q)))$	$p(qe) = ((pe)e)(((qe)e)e)$
$p \vee q . \& . p \vee \neg q = p$	$((p \ q)(p \ (q))) = p$	$(p(qe))((p((qe)e))e) = p$
$p \vee . q \& r = p \vee q . \& . p \vee r$	$p \ ((q)(r)) = ((p \ q)(p \ r))$	$((qe)((re)e)(pU)$ $= (p(qe))((p(re))e)$

Table 10.5: Boolean Algebra as PUT Functions

Algebraic demonstration using the PUT quasigroup can also be applied to traditional logical deduction. The iconic parens form is far more efficient than the algebra of PUT functions, however demonstration of logical deduction using quasigroups is proof in principle that logic can be formulated without calling upon any of the operators of Boolean algebra.

As a demonstration of PUT logic, the Subsumption theorem is proved (Subsumption is also presented in Section A.6). The PUT identities are listed in Table 8.3.

SUBSUMPTION  $((p) (p \ q)) = p$

$(pe)((p(qe))e)$	to reduce to p	
$(p(qe))((pe)e)$	Indep	$a(bu)=b(au)$ $a/pe, b/p(qe), u/e$
$((pe)(p(qe)))((pe)e)$	+Per	$v(au)=(av)(au)$ $a/pe, v/p(qe), u/e$
$(e(p(qe)))((pe)e)$	-Per	$(av)(au)=v(au)$ $a/p, v/e, u/qe$
$(pe)e$	-Occ	$eu=\emptyset$ $e/e, u/p(qe)$
$p$	-Inv	$(ae)e=a$ $a/p, e/e$

Relative to the iconic proof, the functional proof is burdened by ordering constraints. Thus, Independence must be applied to prearrange the function string for an application of Pervasion.

Next, modus ponens is shown to be TRUE (modus ponens is also presented in Section A.4).

MODUS PONENS  $((((p)((p) \ q))) \ q)$

$((((pe)((pe)(qe))e))e)(qU)$	to reduce to e	
$((((pe)(qe))((pe)e))e)(qU)$	Indep	$a(bu)=b(au)$ $a/(pe)(qe), b/pe, u/e$
$((((qe)((pe)e))e)(qU)$	-Per	$(av)(au)=v(au)$ $a/pe, v/qe, u/e$
$(q(((qe)((pe)e))e))(qU)$	+Per	$v(au)=(av)(au)$ $a/q, v/((qe)((pe)e))e, u/U$
$((((qe)((pe)e))(qe))(qU)$	Indep	$a(bu)=b(au)$ $a/q, b/(qe)((pe)e), u/e$
$((q((qe)((pe)e))(qe))(qU)$	+Per	$v(au)=(av)(au)$ $a/q, v/(qe)((pe)e), u/e$
$((((qe)(q((pe)e))(qe))(qU)$	Indep	$a(bu)=b(au)$ $a/q, b/qe, u/(pe)e$
$((e(q((pe)e))(qe))(qU)$	-Per	$(av)(au)=v(au)$ $a/q, v/e, u/(pe)e$
$(qe)(qU)$	-Occ	$eu=\emptyset$ $e/e, u/q((pe)e)$
$e(qU)$	-Per	$(av)(au)=v(au)$ $a/q, v/e, u/U$
$q(eU)$	Indep	$a(bu)=b(au)$ $a/e, b/q, u/U$
$eU$	Dom	$a(eu) = eu$ $a/q, e/e, u/U$

Interpret  $eU$  as  $[(\ )]$ , which is the parens form for TRUE.

Relative to the iconic proof, the functional proof is burdened both by ordering and by arity constraints that obscure the application of Involution. The iconic form can reduce  $((A \ B))$  to  $A \ B$ , but without a grouping operation, the multiple result  $A \ B$  is cannot be expressed functionally. Thus a longer route of proof is necessary, including several ordering and constructive Pervasion steps. The emphasis on left- and right- operations in algebraic manipulation can be understood as limitations imposed by string-based rules that enforce ordering and grouping. The iconic formalism demonstrates that these rules are not essential. For comparison, the iconic interpretation of the functional transformations is presented in Table 10.6.



FUNCTIONAL	OPERATION	ICONIC	INDUCED
$((((pe)((pe)(qe))e))e)(qU)$		$( ((p)((p) q))) q$	
$(((((pe)(qe))((pe)e))e)(qU)$	Indep	$( (((p) q)(p))) q$	yes
$((((qe)((pe)e))e)(qU)$	-Per	$( (( q)(p))) q$	
$(q(((qe)((pe)e))e))(qU)$	+Per	$(q (( q)(p))) q$	yes
$((((qe)((pe)e))(qe))(qU)$	Indep	$(( (q) (p)) q) q$	yes
$((q((qe)((pe)e)))(qe))(qU)$	+Per	$((q (q) (p)) q) q$	yes
$((((qe)(q((pe)e)))(qe))(qU)$	Indep	$(( (q) q (p)) q) q$	yes
$((e(q((pe)e)))(qe))(qU)$	-Per	$(( ( ) q (p)) q) q$	
$qe(qU)$	-Occ	$( q) q$	
$e(qU)$	-Per	$( ) q$	
$q(eU)$	Indep	$q ( )$	yes
$e$	-Dom	$( )$	

Table 10.6: Functional and Iconic Deductive Efficiency

All applications of Independence are induced by the ordering constraints of the symbolic technique. As well, the one-depth-at-a-time Pervasion steps are induced by binary arity. The iconic proof of modus ponens requires three steps.

$(( (p) ((p) q ) )) q$	initial arrangement
$(p) ((p) q ) q$	Involution
$(p) ( ) q$	Pervasion
$( )$	Dominion

## 10.4 Mappings to Boolean Algebra

LoF has been characterized as "simply another axiomatization of Boolean algebra" [gould72]. We have shown how this interpretation can come about, by adding a null token in place of void and by replacing the independence of container contents by a logical connective. Interpreted as logic, the null token provides a representation for the constant FALSE. The inert space shared by the content of any container is given the connective role of disjunction. Both of these modifications impose symbolic structure on what in LoF is absence of structure (i.e. void). Since the obliteration of void with logical form is common to almost all misinterpretations of LoF, we'll call this particular error the error of induced symbolism.

Spencer Brown provides the map from LoF to logic (Table 10.1) in Appendix 2 of the book, although his listing does not include mapping to the two truth values. So it is not surprising to see scholars identifying LoF with Boolean algebra. Spencer Brown thought that the connection between the LoF algebra and the LoF

arithmetic was a significant contribution. However, truth tables also make this connection, although less elegantly. By using an iconic notation and void-based rules, LoF greatly simplifies logic. Spencer Brown thought this too was a significant contribution.

Boolean algebra, however, cannot be "simplified", it is the algebraic system that it is, consisting of two ground values (0,1), two binary operations (meet, join) and one unary operator (complementation). Assigning to void both the value FALSE and the connective OR does not make LoF into Boolean algebra, it simply shows that by adding to (i.e. by extending) LoF, one can reach an interpretation of LoF as Boolean algebra. It also shows that symbolic algebra has too much mechanism to model iconic computation. Spencer Brown:

"...by the mere principle of avoiding an unnecessary prolixity in the representative form, we make the process of calculation considerably less troublesome."

The modifications made to LoF by scholars as they convert LoF into Boolean algebra are informative, not only in clarifying the relationship between LoF and Boolean algebra, but also in clarifying the ways that symbolic mathematicians must modify an iconic formalism to turn it into a symbolic formalism. Most informative is the attitude of these scholars in protecting the status quo. Many do observe that LoF is different, almost all then claim that the difference is an error. Yes, Spencer Brown could have been more user friendly, but he was not in error.

Four of the retorts to LoF that are mentioned next were written circa 1980, a decade after LoF was first published. Ordinarily, these analyses would be forgotten, but like Peirce's Existential Graphs seventy years earlier, Spencer Brown's iconic algebra was condemned rather than analyzed for what it is. And it appears that the condemnation has dominated perception. Meguire's more recent publication confirms that the academic community is yet to understand Spencer Brown's work.

#### 10.4.1 Banaschewski

In "On G. Spencer Brown's Laws of Form" (1977), Banaschewski [ref] identifies multiple containment with inclusive OR and containment itself with "exponentiation", to which he assigns the logical function exclusive OR (i.e. XOR). Crossing is taken apart in two objects, with the outer mark as the exponentiation operator. Table 10.7 shows Banaschewski's map, mediated by annotated parens notation.

BOOLEAN ALGEBRA	ANNOTATED PARENS	PARENS
$\emptyset$	$[\emptyset]$	
1	$[(\emptyset)_a]$	$(\ )$
$x \text{ OR } y$	$[x \ y]$	$x \ y$
$\text{NOT } x = x \text{ XOR } 1$	$[(x \ ( \ )_d)_b((x)_e(( \ )_g)_f)_c)_a]$	$(x)$
$x \text{ AND } y = \text{NOT}(\text{NOT } x \text{ OR } \text{NOT } y)$	$[(x)_b(y)_c)_a]$	$((x)(y))$

Table 10.7: Banaschewski's Map

Banaschewski establishes isomorphism by presenting look-up tables for the "binary functions" of void and mark, mapping them onto the Boolean functions OR and XOR. From this map Banaschewski concludes:

"... primary algebra is exactly the theory of join and addition (= symmetric difference) of Boolean algebras. The primary arithmetic...may be regarded as an algebraic system consisting of two elements and two binary operations."

The difficulty here is that mark, as a container, is not a binary operator. As an object, the mark is a container that is insensitive to the cardinality of its contents. As an operator, it is an instruction to change spaces, from one side of the distinction it indicates to the other. More problematic, void is not only not an operator, it is also not an object. The meaning of iconic space is at the core of the difference between symbolic and iconic mathematics. " ". How many spaces are between the quotation marks? Should we read this as "space", or as "space OR space", or as "space OR space OR space", or as one of the infinite other options? If this is a mapping to Boolean algebra, does it show that LoF is ambiguous, or that Boolean algebra is redundant?

The root problem is that icons are not symbolic functions. They can be interpreted as functions, indeed they can be interpreted as jars of mayonnaise. Had Spencer Brown intended LoF to semantically align with function theory, one would suppose that he would have said so. Because the book meticulously avoids that interpretation, it is improper to conclude that Spencer Brown was hiding a symbolic interpretation behind iconic images.

Banaschewski took the rather circuitous route of identifying  $(x)$  as  $x \text{ XOR } 1$ , rather than the simpler route, which he acknowledges,  $(x)$  as  $\text{NOT } x$ . He did this, apparently, so that the mark might be identified with a binary operator. He claims a problem with the interpretation of  $(x)$  as negation,

"...from this point of view primary arithmetic and primary algebra have different type, and the former no longer generates, in the sense of generation of equational classes, the latter."

That is to say, Banaschewski's interpretation of the LoF arithmetic does not align with his interpretation of the LoF algebra, given that symbolic arity is projected upon the iconic equations. What is missing is that the different types of the arithmetic and the algebra are artifacts of induced symbolism, not oversights of the original notation.

Banaschewski's transcription of Position, for example, is inconsistent with Spencer Brown's transcription of the same rule.

POSITION (p (p)) =

Spencer-Brown: NOT. p OR NOT p = FALSE

Banaschewski: p XOR 1 . OR p : XOR 1 = 0

Were this interpretation to be reinterpreted back into LoF, the rule of Position would not be regained. Banaschewski's interpretation is of the arrangement

( ( ( (p ( )) ((p)(( ))) p ( )) (((p ( )) ((p)(( ))) p)(( ))) )

Table 10.3 shows the annotated parens form of  $x \text{ XOR } 1$  to highlight the unneeded complexity of this interpretation. This situation is worse than a dubious interpretation, because it overtly contradicts Spencer Brown's guidance, in effect placing no credence in the author's own explanations.

The symbolic mathematicians who write about LoF pay little attention to notational variants. Since  $x \text{ XOR } 1$  reduces to NOT  $x$ , the two expressions are taken as the same "up to isomorphism". However, there is a deeper dynamic, LoF is not only an alternative notation, it is also an alternative conceptual system, one that does not follow the rules of symbolic representation. Banaschewski's perspective is that to understand LoF it is necessary to transcribe its structure. Since notational variation, from the symbolic perspective, has been abstracted so as not to be relevant, induced symbolism is both justified and natural. However as Lou Kauffman, a knot topologist who understands both iconic notation and LoF, writes,

"A change of context and style (notation) may reveal the intuitive and conceptual underpinnings of a field, in a way that other, formally equivalent systems, do not. A simple example is Roman to Arabic numerals."

## 10.4.2 Kohout and Pinkava

Kohout and Pinkava [ref] took an algebraic approach to converting LoF into Boolean algebra in "The Algebraic Structure of Spencer Brown and Varela Calculi" (1980). They

"...concentrate on the formal properties of what are actually syntactic rules over a set of symbols."

The LoF notation is not "actually" composed of symbols, and the rules are not necessarily syntactic. Kohout and Pinkava begin with a misinterpretation and then proceed to analyze LoF as if it were symbolic. Common to this type of error, the authors go to great lengths to fix the LoF notation.

"We are thus able to derive the formal properties of his calculus and align them with conventional symbolics and algebraic systems."

LoF is analyzed as if it were a symbolic algebra, not a logic. But symbolic algebra and logic share the same notational bias, they are both string systems.

"We can immediately see the two peculiarities of Spencer Brown's notation mainly that no distinction is made (a) between the 0-ary and the unary operation, both being denoted by  $\top$ , (b) between the 0-ary and the binary operations, both denoted by "empty space" (blank)..."

From the perspective of the LoF iconic notation, the mark is (unconventionally) both an object and an operator because the concepts of object and operator are themselves not particularly relevant to the calculus. LoF arrangements are spatial patterns, what may be identified as an object is entirely contextual. LoF arrangements are also spatial networks that can be seen to propagate signals. What may be identified as an operator is entirely contextual.

The interpretation that a mark is both a constant and a unary operator is not descriptive of LoF, rather it is descriptive of the interpretation placed upon LoF. The mark is assigned an arity since arity is an essential concept for symbolic algebra. Then that assignment is seen to be problematic, "...the double role of  $\top$  is a bit worrying." One solution to this quandary is to reconsider the original interpretation; perhaps the mark does not exhibit the concept of arity, perhaps it is non-symbolic. Instead Kohout and Pinkava engage in induced symbolism,

"Let us introduce  $\varepsilon$  (blank) for "empty space",  $*$  for the concatenation of two strings in Spencer Brown's calculus, and the symbol  $\top_c$  for  $\top$  in the role of a constant (distinguished element)."

From this, Kohout and Pinkava conclude

"... the Spencer Brown calculus is nothing else but a (functionally complete) two-valued calculus..."

Kauffman provides an alternative perspective:

"each mark in an expression acquires its possible uses as operator or operand through its (spatial) relationship to all of the other marks in the expression. Thus each expression in the primary arithmetic is rightly seen as a whole and as a delicately balanced network of relationships among its component marks (parts). When the primary arithmetic is seen as a whole then various general patterns emerge..."

### 10.4.3 Schwartz

In "Isomorphisms of Spencer-Brown's Laws of Form and Varela's Calculus for Self-reference" (1981), Schwartz [ref] declares,

"The axiomatized propositional calculus PC is "essentially isomorphic" with a rigorously defined representation PA of Spencer Brown's primary algebra."

Isomorphism is a strong criteria, declaring that LoF is simply a notational variant, without new mathematical content. Schwartz is admirably clear,

"In order to present the primary algebra as a rigorously defined formal system, the following amendments are made to Spencer-Brown's original formalism:

- 1) the symbol  $\varepsilon$  is used to denote the blank space,
- 2) square brackets are used to replace the mark  $\top$ , so that expressions may be written as linear strings of symbols rather than two-dimensional arrays,
- 3) the mark  $\top$  by itself, thought of as "covering" a blank space, thus becomes written as  $[\varepsilon]$ ,
- 4) Brown's "rule of dominance" is formalized (as the following axiom 3) to govern the role of  $\varepsilon$ ,
- 5) commutativity and associativity of concatenation are formalized (axioms 4 and 5),
- 6) symmetry and transitivity of equality are formalized as rules of inference."

"Rigorously defined formal system" should itself be amended to read "rigorously defined symbolic formal system", in acknowledgment of Spencer Brown's seminal contribution of an iconic formal system. In mathematical writing, rigorous is a synonym for symbolic, since the two concepts are historically co-defined.

Schwartz' amendments bear some resemblance to the interpretation of LoF marks as relational formulas, but with some clear distinctions.

Amendment 1:  $\emptyset$  does not denote the blank space, it denotes the absence of contents within a container. The distinction is critical. The blank space, often denoted by  $\varepsilon$ , is an element in formal language theory, a theory that addresses the structure of string representations.  $\varepsilon$  occurs between words and in the absence of words. As such,  $\varepsilon$  is a singular symbol. It is possible, for example, to write  $\varepsilon\varepsilon\varepsilon$ , a series of three blank spaces.  $\emptyset\emptyset\emptyset$ , however, is meaningless.  $\emptyset$  is not a symbol nor an icon within LoF, neither is  $\varepsilon$ . But  $\varepsilon$  comes with a predetermined interpretation as a string element. Schwartz' supporting vocabulary confirms his misinterpretation, "the blank space" is a symbol, but empty space is nothing at all.

Amendment 2: Brackets and parentheses are convenient notations for  $\sqcap$ , but care must be taken not to bring the string interpretation of a delimiter into its use as a LoF mark. In particular,  $\sqcap$  is one symbol,  $[$  and  $]$  are two symbols.  $\sqcap$  does not have a right or a left side, the sides of a symbol derive from string sequences.

Amendment 3:  $\sqcap$  does have an inside and an outside, however these two locations do not identify the location of an "inside symbol" or an "outside symbol". The locations identified by a mark are spaces that can accommodate any other arrangements. Therefore, an empty mark does not cover the object called a blank space, it covers empty space.

Void, being philosophically as well as symbolically nothing, does not support any form of syntactic rules or any structural interpretation. Any apparent properties of void, such as being an argument to a function, or creating a relational connectional between forms sharing a space, or being multiple in representation, any property of nothing is a projection from the domain of interpretation, and not within the domain of LoF syntax.

Amendment 4: Schwartz' formalization of the role of  $\varepsilon$  is via the axiom  $\varepsilon e = e$ , where  $e$  is an expression. Again, this places void in a particular location, as if it were an object. Schwartz is objectifying absence rather than formalizing empty space.

Amendment 5: Commutativity and associativity are also appended to the structure of LoF arrangements. Both of these concepts are characteristic of binary

operators, not of spatial containers. Again Kauffman reinforces clearly Spencer Brown's position,

"Here it is not so much that commutativity and associativity are tacitly assumed, but that we are articulating a level at which they do not yet exist!" [ref]

Amendment 6: The final amendment is not an amendment at all. Spencer Brown establishes the symmetry and transitivity of equality within the text. But this topic does require a clarification. When LoF is redefined by assigning logical meaning to empty space, the use of equality is also redefined. On this issue, Schwartz states

"A consequence of this isomorphism result is that "equality" in PA is isomorphic with "logical equivalence" in PC\*."

That is, if one believes that LoF is the same as propositional calculus, then the algebraic use of equality in LoF is the same as logical equivalence. Meguire [ref] has also made this error, a form of begging the question. Orchard [ref] makes the more sophisticated observation, "...two logically equivalent statements are not necessarily identical."

Here then is the error of induced symbolism in microcosm: Turning any system into another system that it is not does not help to clarify what that system is. Even if LoF were isomorphic to Boolean algebra (which it is not), then it would still be different. Isomorphism elects to discard all other characteristics of a system except structural equivalence. But when the structural abstractions themselves are incommensurable, such as the case of recording in different dimensions (symbols as one-dimensional strings, marks as two-dimensional icons), then morphism as a tool is inappropriate. Metaphorically for example, we do even try to establish a morphism between a book and the movie of the book.

It is difficult to ask a professional not to use the tools of the trade, but we do know not to call a plumber to do an electrician's job. Spencer Brown's work is of deep interest because he has crafted a mathematical system that, by the symbolic standards of modern mathematics, is not a (symbolic) mathematical system. LoF is simply a new type of formal system.

#### 10.4.4 Cull and Frank

Collect every published misinterpretation of LoF, and wrap it in an arrogant exposition and you will have the Cull and Frank [ref] article, "Flaws of Form" (1979).



"Brown i[thinks] that he is using only one symbol; he thinks that his language has only one character, the corner, or that if there is a blank in his system, there is only one token of it, simultaneously occupying all the unscripted paper on which equations are written. ... In thinking of his language as containing only one character, Brown violates the most basic truth of information theory: at least two symbols are required to convey any information."

The Shannon theory of information is built upon binary strings. It is not an analysis of the information content of images. LoF is *prima facie* evidence of a system that uses only one token (in space) to communicate information, all that is required is to look at LoF containment arrangements. Cull and Frank have the correct interpretation of pervasive space, "simultaneously occupying all the unscripted paper", but they are then drawn to deny even the possibility of that interpretation.

"Moreover, as we shall see, both the corner and the blank must be available in an indefinitely large number of sizes."

Again correct to a point. Unlike a sequence of tokens, which measures off symbolic distance one character at a time, there is no metric underlying the space upon which marks are recorded. LoF explicitly does not include the concepts of size and number.

"Brown treats blank spaces as infinitely ambiguous"

The core of this sentence approaches Spencer Brown's intention, although technically nearly every word is incorrect. Not blank spaces, but empty space. Not infinitely ambiguous, but capable of supporting any arrangement. The deeper meaning of the absence of a mark is that emptiness permeates the representational substrate (a rather obvious observation). To replace emptiness by a symbol, even the symbol for blank space, would require arbitrarily many symbols. Cull and Frank are viewing the representational space as positional and structured, permitting, for instance, only one correct place to put the blank token. The same principle is also apparent in writing  $v$  for concatenation. Since LoF is a spatial notation, concatenation is not well defined. It is better called "sharing a space", invoking none of structure of a linear, typographical notation. And with sharing space, there is no logical connector that connects objects within the space into pairings.

"Brown merely replaces the ordinary ideographic notation of mathematics with a positional or analytic notation."

Here, Cull and Frank need an editor. Ideographic is a synonym for iconic, while positional is synonymous with string-based. The authors have their definitions mixed up. Next, the common thread across these misinterpretations,

"Before considering his axioms, we will exchange Brown's notation for a more conventional one.... From a mathematical point of view, notation is insubstantial...it cannot change the subject the language is used to inform us about."

That is, of course, unless the notation is the subject. One additional word to fix the problem: "symbolic notation is insubstantial". Cull and Frank's symbolic conversion is yet another variation of the same idea that void cannot be allowed to stand for itself,

"By allowing "1" to replace  $\sqcap$ , "0" to replace the blank, "v" for concatenation, and " $\oplus$ " for superimposition, the axioms appear as  
 $1 \vee 1 = 1$  [and]  $1 \oplus 1 = 0$ "

The theme of induced symbolism is by now well understood. By turning iconic forms into string forms, an iconic formalism can be converted into a symbolic formalism. The pattern maker  $\sqcap$  is converted into a character symbol; pervasive space is first converted into the blank character and then linearized by placing a connective symbol between each of the newly constructed icons-as-symbols. Spatial containment is converted into a notation for exponentiation. Once the conversion is complete,

"Brown's system is the same as any other system of axioms employing logical symbols with the same meaning..."

Spencer Brown's system becomes the same as other symbolic systems precisely because it was converted into a symbolic system. What of the characteristics of the iconic system that were lost? The computational efficiency, the parallelism, the deep rules, the notational variety as blocks and paths and graphs, are these differences worth discarding so that mathematics does not grow into new territories? Or is it possible to acknowledge that string-based mathematical formalism has weaknesses to be addressed by new techniques?

#### 10.4.5 Meguire

#### 10.4.6 Not Boolean Algebra

Of course, the token FALSE is not equivalent to no representation at all, and disjunction is not equivalent to independence. There are two perspectives at work here. From the perspective that LoF does indeed incorporate (implicitly) conventional logic or conventional algebra, then the independence of contents never did exist, and the failure to label the concept FALSE is an oversight that makes LoF, well, more confusing. This is not Spencer Brown's intention, and has no support within the text or from within the mathematical system.

The other perspective is that LoF incorporates seminal new mathematical ideas. LoF does map one-to-many onto propositional logic, reducing the redundancy within logical notation. It also provides new methods of problem solving and new modes of logical thinking. Although we sympathize that it is indeed difficult to learn these new ideas, failure to distinguish between words and images seems insufficient justification to publish incorrect interpretations of Spencer Brown's work.

However, in a wider sense, none of these authors are to blame. They are merely expressing what they have been taught, that the pinnacle of mathematical evolution, initiated by Hilbert a century ago, is to remove the frailty of human intuition from formal thought by strictly separating what is written from what is meant. Notation is not to be looked at, it is to be blindly interpreted.

Stated again, Hilbert's Program begins to sound like a bad idea: the pinnacle of mathematical evolution is to have what is written bear no resemblance to what is meant.

By following the notational consequences of an interpretation of LoF as a calculus of containers, we arrived at a single relation, Contains, that both adequately represents the intention of LoF and adequately meets the criterion of string-based processing. The Contains relation, in turn, can be expressed functionally as the PUT function. If indeed LoF is simply another notation for Boolean algebra, then Boolean algebra itself is described by a single function. The position that LoF is isomorphic to propositional calculus and to Boolean algebra has a downside, since it implies that the properties of LoF are also the properties of conventional mathematics. It is surely safer to disavow isomorphism between iconic and symbolic systems than to reduce both to a common denominator. But it is difficult to believe that the intent of these articles, with the exception of Meguire, was to inform or to clarify, since what they overtly do is to condemn innovation without understanding or analysis.

Here are several simple facets of LoF, each of which provides a template with which to prove that LoF is not isomorphic to Boolean algebra:

- Legitimacy of Nothing: LoF has one ground object, while Boolean algebra has two.
- Non-functionality: LoF does not participate in the theory of groups since there are no functions in the theory.
- One Relation Only: LoF maps to a single Contains relation, while Boolean algebra requires two functions.
- Not One-to-One: The map from LoF to logic is one-to-many.
- Not Symmetrical: The equivalence class of marked LoF arrangements is approximately twice as large as the equivalence class of void arrangements.

## 10.5 Asymmetric Duality

We will count how many parens arrangements reduce to each of the two equivalence classes generated by the arithmetic of LoF. Exactly one-half of all possible logical expressions reduce to TRUE, and one-half to FALSE, due to the duality and symmetry of logical form.

"...the criterion for whether a system is complete and consistent is simply that exactly half the possible strings of a given length are eventually generated if one starts from the string representing "true"." [wolfram p797]

Wolfram's quote emphasizes that almost all of what we know about mathematical structure rests on the presumption that mathematical concepts are represented as strings.

Counting the number of parens arrangements is a well understood problem. The Catalan numbers count how many ways  $n$  pairs of parentheses can be assembled. For example, there are five different ways that three parentheses can be arranged

(( ( )))    (( ( ))    (( )) (    (( )) (    (( )) (

The Catalan numbers do not count the number of LoF parens arrangements, because different sequential orderings (the third and fourth figures above, for example) are counted in the Catalan sequence. Catalan numbers count the number of ordered trees. Parens arrangements, in contrast, are insensitive to ordering, they are counted by the number of unordered trees, commonly called rooted trees. There are four rather than five unordered trees in the example above.

A rooted tree has a single starting node called the root. This root is the universal container of parens arrangements. Another difference between the Catalan sequence and the rooted tree sequence is that rooted trees require one additional container to serve as the root. The rooted tree sequence is therefore offset by one from the Catalan sequence, since one additional parenthesis is absorbed by the root. Table 10.8 shows the Catalan and rooted tree sequences for a given number of parentheses (i.e. containers), together with illustrations of the first few rooted tree configurations. Table 10.8 also shows the number of rooted tree arrangements that reduce to each of the LoF equivalence classes.

PARENS	CATALAN	ROOTED	EXAMPLES	=[]	=[()]
1	1	1	[]	1	0
2	2	1	[()]	0	1
3	5	2	[()] []	1	1
4	14	4	[()] [()] [()] [()]	1	3
5	42	9	[()] [()] [()] [()] [()]		
6	132	20	[()] [()] [()] [()] [()] [()]	4	5
7	429	48	[()] [()] [()] [()] [()] [()] [()]	19	29
8	1430	115	[()] [()] [()] [()] [()] [()] [()] [()]	38	77
9	4862	286	[()] [()] [()] [()] [()] [()] [()] [()] [()]	107	179
10	16796	719	[()] [()] [()] [()] [()] [()] [()] [()] [()] [()]	247	472

Table 10.8: Catalan and Rooted Tree Sequences

The rooted tree sequence begins with the two simple parens forms,  $[]$  and  $[(C)]$ . Three parens generate two different arrangements, corresponding to the right-side of the Crossing and Calling rules. When rooted trees are reduced by the LoF arithmetic, the number of trees that reduce to mark exceeds the number that reduce to void by a ratio of approximately 2 to 1. This should not be surprising since Dominion reduces all arrangements with a mark in U to the mark equivalence class.

Figure 10.3 graphs the ratio of parens arrangements that reduce to the mark equivalence class vs the number of parens/containers in the arrangement.

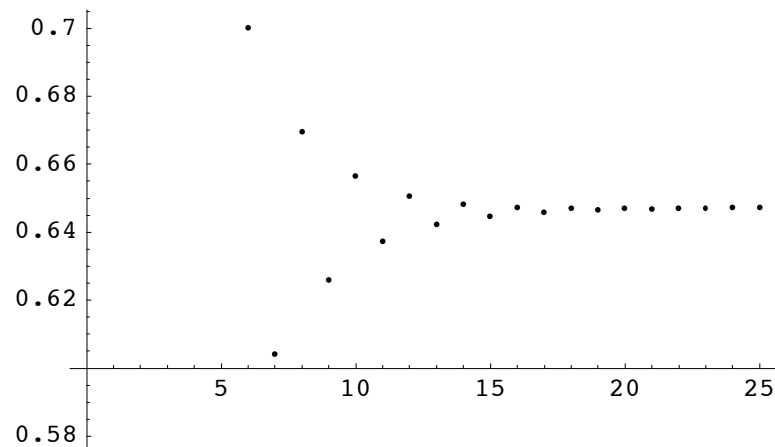


Figure 10.3: Equivalence Classes of Parens Arrangements

Figure 10.3 shows that as the number of containers in an arrangement increases, the percent of these arrangements that reduce to mark converges to approximately 64.7%.

When LoF is interpreted as propositional logic, the result is remarkable. LoF is a biased logic, there are more TRUE arrangements than FALSE arrangements. In terms of void-equivalence, ~35.3% of all arrangements are void-equivalent.

Analysis of equivalence classes is conventionally considered to be an aspect of the semantics of a notation. The bias in LoF reduction addresses what arrangements mean. Since the LoF equivalence classes do not align with those of propositional logic, LoF cannot be morphic to propositional logic. This result provides a measurement of the impact of converting the iconic formal system into a symbolic formal system (by assigning logical interpretations of FALSE and OR to void). In particular, the technique of induced symbolism changes the equivalence class of ~14.7% of LoF arrangements.

## 10.6 Cognitive Interpretations

We brief mention a few new cognitive perspectives that arise from treating logic iconically.

The map from logic to LoF associates the logical constant FALSE with the LoF concept of void. Within LoF there is no representation of FALSE. Here then is a logic that does not acknowledge the existence of logical duality in thought. True ideas are confounded with existent, relevant structure. False ideas are discarded as irrelevant. Certainly existent arrangements can be constructed that reduce, via rules, to nothing at all. These arrangements are illusions, clutter in the way of clear thinking. This can be expressed as a paraphrase of

the Principle of Void-Equivalence: Structure that is FALSE is semantically inert and syntactically irrelevant. That is, FALSE structure has no impact on clear thinking.

LoF rules involve deletion and construction of images, rather than rearrangement of strings. Logical deduction currently rests upon strategies of accumulation of facts, rearrangement of collections of facts, and strategic planning to assemble and rearrange facts to arrive at conclusions. The techniques of modus ponens, disjunctive syllogism, reductio ad absurdum, etc. are ancient artifacts that still drive the organization of logical proof. None are particularly relevant to the direct iconic formulation. Rational thought might instead be seen as selective forgetting of irrelevancies, rather than as the accumulation and correlation of potentially relevant facts.

The word OR has both inclusive and exclusive uses in common language. Logicians define the connective OR as inclusive: one or the other or both. LoF represents disjunction by placing arguments within the same contained space, very much like people all sharing a room together. Viewing one or another or all together conforms naturally to this image. Disjunction is often identified with choice. The iconic perspective makes the items of choice conceptually independent, and places the connection not between items but between each item and the chooser. The iconic representation of exclusive OR directly shows the structure of comparison:  $((a\ b)((a)(b)))$ .

Spencer Brown's point that associativity, commutativity, and arity are not central to the understanding of logic comes also with a more powerful frame of mind. It does not matter how many people share a room, nor is there any prerequisite ordering or grouping of those people. Here logical thinking is freed from linear structuralism, and placed firmly within spatial visualization.

The logical connectives themselves lack an adequate visual reference. The grammar of ancient Greece first acknowledged syncategormata, words in language that do not refer to any actual thing. Just what object does a logical connective such as IMPLIES identify? LoF, and its notational varieties, now provide visual images of the structure of logic in language. Implication is the separation of content and context by a boundary. Implication is also stacking blocks, crossing a boundary, traversing a path, and making a distinction.

Logical implication is not as well established in language as are the Boolean operators NOT, OR, and AND. Training in logic often focuses on implication, for both historical and deductive reasons. When we focus on the boundary between two arrangements, such as (1) 2, we might say that Object-1 is-Distinguished-from Object-2. This is equivalent (at least structurally) to saying that Object-1 IMPLIES Object-2. Additional metaphors and visualization of implication cannot but help our understanding of deduction.

Deep Pervasion changes the idea of implication more profoundly. What was once a chain of implications, say in the form of

((((1) 2) 3) 1:                    1 IMPLIES 2 . IMPLIES 3 : IMPLIES 1

efficiently reduces to 3 IMPLIES 1,

((((1) 2) 3) 1	
(( ( ) 2) 3) 1	deep pervasion
(                3) 1	occlusion

That is, nested replications can all be eliminated from logical consideration.

Within the logical interpretation of LoF containers, crossing inwards does not change the value of an arrangement, while crossing outwards does change the value. That's the asymmetry at the heart of conventional logic, the semipermeability of distinction -- together with the illusory duality of TRUE/FALSE. LoF is a logic without opposites of any kind, and that's quite a revolutionary step for Spencer Brown (and for C.S. Peirce before him) to have taken. By providing a formal system built upon a single concept, that can be interpreted as logic, they have freed both argumentation and computation from enforced and exclusionary dualism. Metaphysically, Kashmir Shaivism holds that maya (illusion) is duality, not the material world itself.

Finally, Spencer Brown's foundational concept is that of distinction. The container represents a distinction between inside and outside. Distinction is intended to draw attention purely to difference. Rather than thinking of distinction as a representation of logical truth, we might consider logical truth to be a facet of distinction, making distinction more primitive and more fundamental than logic and truth, a firmer fundament upon which to build mathematical systems. Truth is not a prerequisite for value, it is a facade that is necessary to make sense out of the meaningless of symbols.

## 11 Conclusion

In the conclusion we offer a brief summary of what might lead to a theory of representation. We also address the misinterpretations of Spencer Brown's Laws of Form.



## 11.1 A Theory of Representation

Symbolic mathematics holds that notation is completely separated from meaning, so much so that the particular characteristics of a notation do not interact with the particular characteristics of the mathematics expressed by that notation. So long as notation itself is conceptualized as being built from the concatenation of tokens to form strings, this may be the case. However, it is also clear that in the 21st century, information and meaning are conveyed by methods far more diverse than the written and spoken word. We can find no reason to believe that the expression of mathematical concepts must be limited to strings of symbols. In fact the belief that meaning can be separate from the mode of recording that conveys that meaning itself suggests that symbolic strings have no privileged status. This is clearly not the case.

We have called the artificial forcing of symbolic representation on abstract concepts, the error of induced symbolism. String-based representation was adopted as a method to enforce rigor. Spencer Brown's and Peirce's iconic systems provide counter-examples, both convey the rigor of mathematics through arrangements of images.

## 11.2 Laws of Form

Laws of Form has been both widely and deeply misinterpreted by mathematicians, and has confused non-mathematicians. We have attempted to highlight some of the reasons that this text has encountered such difficulty in comprehension and in acceptance.

- LoF is perversely obscure.

Spencer Brown has constructed a mathematics that does not rest upon the traditional foundations of logic, set theory and whole numbers. It is not surprising that a reader who is looking for these foundations would be frustrated. Many readers then set out to provide the traditional foundations, finding places where LoF can be converted into sets and logic. We too have taken this path, showing just what is necessary to express LoF within a conventional relational notation. And what is necessary is to modify the essential conventions and structure of the iconic formalism, in the process losing its insights. The innovations within LoF include:

- a formal calculus that does not require sets, logic, or integers
- computation over arrangements of images rather than symbols
- the union of meaning and representation
- the absence of concepts of arity, commutativity and associativity
- use of void with strictly no properties

- strict independence of objects sharing a container
- transformation rules that delete irrelevant structure
- pattern-variables that bind to zero, one, or many arrangements
- the dominant use of void-substitution as computation\*
- parallel rather than sequential transformation steps\*
- path variables, deep rules, and transparency of nesting\*
- geometric generation of alternative notations\*
- vision as a mode of interaction with mathematical ideas\*.

The last five innovations marked by an asterisk are extensions based on LoF, but not mentioned explicitly in the book.

- LoF is isomorphic with Boolean algebra.

An extreme of the "where's the foundations?" position, is finding them so fully that LoF becomes identical to Boolean algebra. The several published proofs of this isomorphism all first change LoF into something that it is not, and then observe that the something that it is not is in fact Boolean algebra. We have attempted to localize the source of the error as induced symbolism. By assigning empty space a meaning (as both the ground value FALSE and the operation OR), the misinterpretation of nothing as a logical something is reminiscent of the intolerance of zero as a concept at the end of the Dark Ages. Dominating this type of error is the step "Let us represent absence by a token." The token is, of course, a negation of absence, rendering this approach as exactly incorrect.

- LoF is mystical.

Although Spencer Brown does engage in psychological and metaphysical analogies in the Notes to LoF, and in his other writings, the text itself is strictly and rigorously mathematical. This complaint can easily be addressed as well to the 99% of mathematicians who are Platonists, seeing mathematics as the eternal inhabitant of a virtual world that is perfect in its abstraction, and that supports impossible structures such as Cantorian infinities.

The purpose of this essay has been to recognize LoF for what it is, rather than for what it is not. In explicating the features of the LoF iconic algebra, we hope to make a small contribution to the growth of mathematical ideas. This evolutionary step centers around readmitting the human body into the pantheon of form. The dominance of symbolic notation in mathematics and in logic has certainly increased professional rigor, but it has also unfortunately increased common discomfort with the representation and use of rigorous thought processes. We believe that providing an iconic option for mathematical thinking may

potentially lead to clearer thinking by non-professionals. It will certainly provide additional tools for thought.

## 12 References

## APPENDIX                      Examples of LoF Reduction via Pattern-Matching

The following examples show LoF rule application through pattern-matching. The examples compare pattern-matching of iconic forms to pattern-matching of relational symbolic forms. Parallel graph reductions are also shown.

Transformation of relational structures incorporates two regimes: explicit pairs, (a,b), and null token pairs, (a,∅). A rule application succeeds whenever every explicit pair in a rule is matched by a pair in the descriptive set of the target arrangement. A rule fails whenever one or more explicit pairs in a rule fails to match the target arrangement. In the case of the null token pairs, a pattern-match succeeds whenever all explicit pairs have been matched and the null token pairs are matched exactly. A pattern-match fails whenever all explicit pairs have been matched, the first member of a null token pair is also matched, and the second null token member of the pair is not matched.

Table A1 shows the parens and the ordered pairs forms of each of the transformation rules used in these examples.

---

CROSSING	$(( )) =$	$\{(u,a),(a,b),(a,\emptyset),(b,\emptyset)\} = \{ \}$
INVOLUTION	$((a)) = a$	$\{(u,b),(b,c),(b,\emptyset),(c,y_-)\} = \{(u,y_-)\}$
DOMINION	$a ( ) = ( )$	$\{(u,x_-),(u,b),(b,\emptyset)\} = \{(u,b),(b,\emptyset)\}$
OCCLUSION	$(a ( )) =$	$\{(u,b),(b,y_-),(b,c),(c,\emptyset)\} = \{ \}$
PERVASION	$a (a b) = a (b)$	$\{(u,y_-),(u,c),(c,y_-),(c,z_-)\} = \{(u,y_-),(u,c),(c,z_-)\}$

---

Table A1: Parens and Ordered Pair Notations for the LoF Transformation Rules

### A.1 Pattern-Matching in the Arithmetic

This example is encoded into ordered pairs in Section 3.5. The example illustrates reduction by two sequential applications of Crossing. Reduction in iconic parens notation is shown first, then reduction in ordered pairs notation. For this example only, failed pattern-matches during search for rule applications are also shown.

---

$( ) ( ( ) ( ( ) ) )$	initial arrangement
-----------------------	---------------------

	( ( ) )	match Crossing
( ) ( ( ) )		apply Crossing
	( ( ) )	match Crossing
( )		apply Crossing

Annotated Parens:  $[(\emptyset)_1 ((\emptyset)_3 ((\emptyset)_5)_4)_2]$

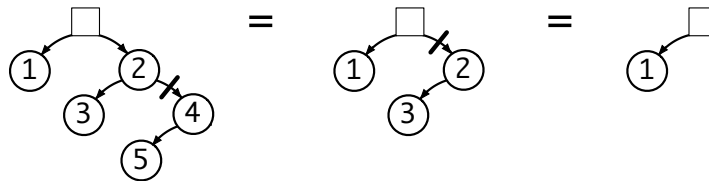
Ordered Pairs:  $\{(U,1),(U,2),(1,\emptyset),(2,3),(2,4),(3,\emptyset),(4,5),(5,\emptyset)\}$

$\{(U,1),(U,2),(1,\emptyset),(2,3),(2,4),(3,\emptyset),(4,5),(5,\emptyset)\}$	initial arrangement
(u,a) (a,b)	fail (a,b)
(u,a) (a,b) (a, $\emptyset$ ) (b, $\emptyset$ )	fail (a, $\emptyset$ )
(u,a) (a,b) (a, $\emptyset$ ) (b, $\emptyset$ )	fail (a,b)
(u,a) (a,b) (b, $\emptyset$ ) (a, $\emptyset$ )	match u/2,a/4,b/5
$\{(U,1),(U,2),(1,\emptyset),(2,3), (3,\emptyset) \}$	apply Crossing
(u,a) (a,b) (b, $\emptyset$ ) (a, $\emptyset$ )	fail (a,b)
(u,a) (a,b) (b, $\emptyset$ ) (a, $\emptyset$ )	match u/U,a/2,b/3
$\{(U,1), (1,\emptyset) \}$	apply Crossing

Result:  $[(\emptyset)_1]$

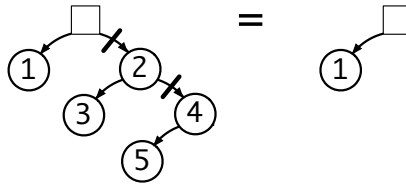
---

The graph reduction shows two applications of the rule of Crossing,



Graph reduction exhibits strong parallelism by proceeding locally, without global coordination. Concurrently, each node checks its lower links. Only in the case of no lower links (Node-1, Node-3 and Node-5 here), a node sends a deletion request to its parent node (Node-2 and Node-4 respectively, U does not participate in rules or reductions). A parent node that receives a deletion request checks its lower links. Only in the case of exactly one lower link, the parent node deletes itself (Node-4 initially). As clean-up, the deleting node instructs its lower node to also delete itself, and instructs its upper node to disconnect (represented by the heavy bar). Here, Node-4 instructs Node-5 to delete, and Node-2 to disconnect. The cycle repeats, here Node-2 now qualifies for deletion.

The algebraic Occlusion rule, which combines Crossing and Calling, is more efficient,



Occlusion triggers twice, once as a disconnect between Node-4 and Node-2, and once between Node-2 and U. Parallelism requires that these two applications do not interfere with each other. Void-based computation in the graph representation manifests as the temporal independence of disconnects. The graph configuration is insensitive to which disconnect occurs first. Rearrangement strategies do not support parallelism because rule applications must wait until the rearranged configuration is in place prior to proceeding.

## A.2 Use of the Pattern-Variable

In this example, the pattern-variable within the Involution rule binds to many objects. The example also includes incidental structure that is not involved in the rule application.

---

1 ( 2 ((3 4)) )	initial arrangement
(( a ))	match a/3 4
1 ( 2    3 4    )	apply Involution

Annotated Parens: [1 (2 ((3 4)<sub>7</sub>)<sub>6</sub>)<sub>5</sub>]

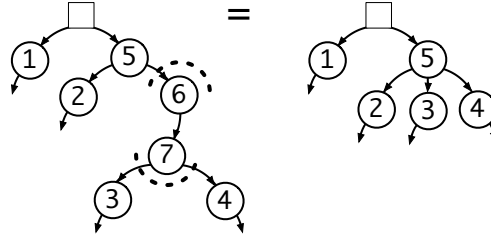
Ordered Pairs:     {(U,1),(U,5),(5,2),(5,6),(6,7),(7,3),(7,4)}

{(U,1),(U,5),(5,2),(5,6),(6,7),(7,3),(7,4)}	initial arrangement
(u,b) (b,c) (c,y <sub>-</sub> )(c,y <sub>-</sub> ) (b,∅)	match u/6,b/7,c/8,y <sub>-</sub> /3 4
{(U,1),(U,5),(5,2),                    (5,3),(5,4)}	apply Involution

Result:                 [1 (2 3 4)<sub>5</sub>]

---

The graph reduction follows,



The nodes participating in Involution are highlighted in the illustration. Each node checks its lower links. Only in the case of exactly one lower link, a node instructs its single child node to transfer its lower links (the grand-children so to speak) upward. Here, Node-6 has one lower connection, it sends a message to Node-7 to pass its children (Node-3 and Node-4) upward. Node-6 then sends the grand-children links to Node-5, and the clean-up message to Node-7. When the grand-children links have been reestablished, Node-5 disconnects Node-6.

The issue of grouping Node-3 and Node-4 as a set, or as pattern-variable bindings, does not occur in the graph implementation. Node-7 transfers its lower links to Node-5 regardless of cardinality. A separate transfer can occur for each node that is being transferred, grouping is unnecessary.

### A.3 Partitioning by Pattern-Variables

In this example, the Pervasion rule applies to a collection of objects. The two pattern-variables partition the objects contained by the outer parens into two collections, those that can be deleted and those that cannot.

---

(1 2 3 4 (3 4 5 6))	initial arrangement
a ( a    b )	match a/3 4,b/5 6
(1 2 3 4 (    5 6))	apply Pervasion

Annotated Parens: [(1 2 3 4 (3 4 5 6)<sub>8</sub>)<sub>7</sub>]

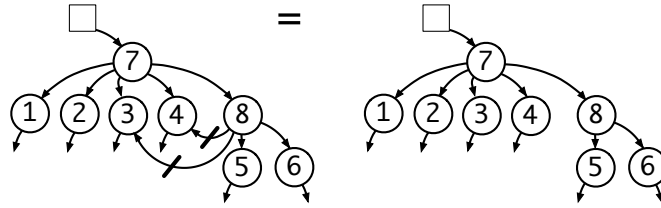
Ordered Pairs:    {(U,7),(7,1),(7,2),(7,3),(7,4),(7,8),(8,3),(8,4),(8,5),(8,6)}

{(U,7),(7,1),(7,2),(7,3),(7,4),(7,8),(8,3),(8,4),(8,5),(8,6)}	initial arrangement
(u,y_)(u,y_)(u,c) (c,y_)(c,y_)(c,z_)(c,z_)	match u/7,c/8,y_/3 4,z_/5 6
{(U,7),(7,1),(7,2),(7,3),(7,4),(7,8),	(8,5),(8,6)} apply Pervasion

Result:            [(1 2 3 4 (5 6)<sub>8</sub>)<sub>7</sub>]

---

The graph reduction follows,



To implement Pervasion, all graph nodes propagate messages downward; each message identifies the children nodes of the node sending the message. As a message descends, the receiving node matches its own children nodes to the nodes identified by the message. Here, Node-7 sends to Node-8 that it is connected to five children nodes, including Node-3 and Node-4. Node-8 identifies that it is also connected to Node-3 and Node-4. When a match occurs, the receiving node disconnects the matched node(s). Propagating messages down the directed graph implements Deep Pervasion.

#### A.4 Proof of Modus Ponens

The predicate calculus inference rule of modus ponens is transcribed into parens using the LoF map between logic and parens. The resulting arrangement is then reduced. The empty container that remains after simplification is interpreted as TRUE. The initial arrangement is encoded using variables rather than concrete numerals to maintain the generality of the modus ponens rule; capital letters are used for these logical variables.

Modus ponens:  $P \text{ AND } . P \text{ IMPLIES } Q : \text{ IMPLIES } Q \quad (((P)((P) Q))) Q$

Modus ponens is quite general, it is intended to apply to simple propositions (represented herein by numerals), to arbitrary formulas or sentences (represented by pattern-variables as capital letters), and to arbitrary collections of sentences (also represented by pattern-variables). Predicate calculus requires collections of sentences to be represented as sets, grafting a separate layer of mechanism (set theory) onto the logical infrastructure. The distinction between propositions, formulas, and sets of formulas is necessary within predicate calculus because of imposed notational restrictions on the structure of a formula. Pattern-variables combine all three conventional types, showing that iconic structures do not require the conceptual articulation that is built into symbolic logic. From the perspective of iconic structure, these articulations are notational artifacts. From the perspective of conventional logic, these articulations have been intimately involved in the definition and growth of logical structure throughout the twentieth century.



(((P ) ((P ) Q ))) Q	initial arrangement
(( a ))	match a/(P)((P)Q)
(P ) ((P ) Q ) Q	apply Involution
( b a ) a	match a/Q,b/(P)
(P ) ((P ) ) Q	apply Pervasion
((a) )	match a/P
(P ) P Q	apply Involution
(a b) a	match a/P,b/∅
( ) P Q	apply Pervasion
( ) a	match a/P Q
( )	apply Dominion

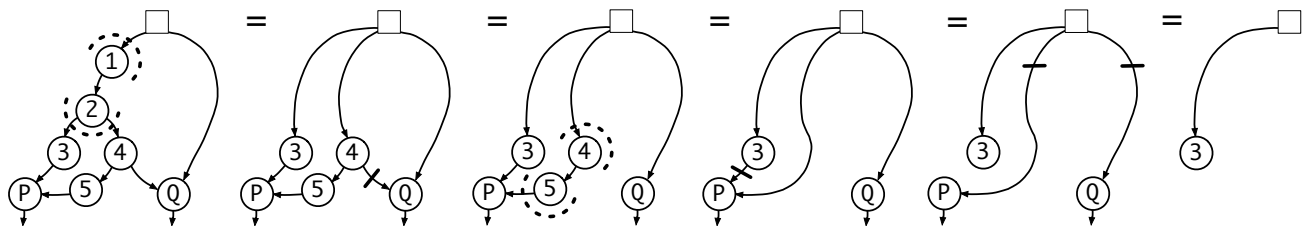
Annotated Parens: [(((P)<sub>3</sub> ((P)<sub>5</sub> Q)<sub>4</sub>)<sub>2</sub>)<sub>1</sub> Q]

Ordered Pairs: {(U,1),(U,Q),(1,2),(2,3),(2,4),(3,P),(4,5),(4,Q),(5,P)}

{(U,1),(U,Q),(1,2),(2,3),(2,4),(3,P),(4,5),(4,Q),(5,P)}	initial arrangement
(u,b) (b,c) (c,y_)(c,y_-) (b,∅)	match u/U,b/1,c/2,y_/3 4
{(U,Q), (U,3),(U,4),(3,P),(4,5),(4,Q),(5,P)}	apply Involution
(u,y_-) (u,c) (c,z_-)(c,y_-)	match u/U,c/4,y_/Q,z_/5
{(U,Q), (U,3),(U,4),(3,P),(4,5), (5,P)}	apply Pervasion
(u,b) (b,c) (c,y_-) (b,∅)	match u/U,b/4,c/5,y_/P
{(U,Q), (U,3), (3,P), (U,P)}	apply Involution
(u,c) (c,y_-) (u,y_-) (c,z_-)	match u/U,c/3,y_/P,z_/∅
{(U,Q), (U,3), (U,P) (3,∅)}	apply Pervasion
(u,x_-) (u,b) (u,x_-) (b,∅)	match u/U,b/3,x_/P Q
{(U,3), (3,∅)}	apply Dominion

Result: [ (∅)<sub>3</sub> ]

The graph reduction that corresponds to the stepwise reduction above is



The first image shows the first reduction step, an application of Involution. The second image implements the application of Involution, and shows the next step, and application of Pervasion. This is followed by another application of Involution and another application of Pervasion. The final step shows the application of Dominion. The resulting single node is interpreted as TRUE.

This graph reduction does not incorporate the natural parallelism available within the graph notation, see the next Section for a graph parallel reduction.

When matching rules are extended by structure sharing, e.g.  $(P)_3=(P)_5$ , Pervasion can be applied to compound arrangements, here (P). This leads to the following iconic reduction:

$((((P) ((P) Q )))) Q$	initial arrangement
$(( \quad a \quad ))$	match $a/(P)((P)Q)$
$(P) ((P) Q ) \quad Q$	apply Involution
$a ( \quad a \quad b ) \quad a$	match $a/(P) Q, b/\emptyset$
$(P) ( \quad ) \quad Q$	apply Pervasion
$a ( \quad ) \quad a$	match $a/(P) Q$
$( \quad )$	apply Dominion

The replication of Pattern-variable-a above is an artifact of recording on a textual line.

#### A.4.1 Cognitive Interpretations

Modus ponendo ponens has been a primary rule of inference in classical logic for millennia. The Latin means "mode that affirms by affirming". One possible interpretation is simply that modus ponens is TRUE by assumption. Another is that modus ponens is built into the definition of what is meant by inference.

Modus ponens:  $P \text{ AND } . P \text{ IMPLIES } Q : \text{ IMPLIES } Q \quad (((P)((P) Q))) Q$

The premise of the rule consists of two parts joined in conjunction (a structure labeled P, and an implication that P IMPLIES Q). The rule says that the conjunction implies Q.

The iconic proof of modus ponens rests upon a different set of assumptions, the three void-based rules of Dominion, Involution, and Pervasion. Were the mode of thought associated with the iconic proof to be described, it might go as follows:

The initial arrangement unnecessarily separates the premises and the conclusion by a double-container. Since double-containers are irrelevant, this one can be deleted (Involution). We are left with two arrangements, (P) and Q, that are both outside of a container and also replicated within that container. Since nested replications are irrelevant, the form (P) Q within the container is irrelevant and can be deleted (Pervasion). What remains is a dominant form, ( ), that renders all other forms in its context irrelevant. These other forms can all be deleted (Dominion). The remaining dominant form is interrupted as TRUE.

In this cognitive reconstruction of the LoF algebra, the concepts of implication and truth do not occur. The entire dynamic is simply to eliminate what is known (by rule) to be irrelevant to the initial arrangement.

## A.5 Parallel Proof of Modus Ponens

Iconic notation incorporates natural parallelism. The contents of any container can be transformed concurrently since content arrangements are, by construction, mutually independent. The proof of modus ponens is repeated, incorporating multiple parallel substitutions during the proof. The parallel proof requires two computational steps, whereas the sequential proof requires five steps.

---

(((P) ((P) Q ))) Q	initial arrangement
(( a ))	match a/(P)((P)Q)
( b a ) a	match a/Q,b/(P)
a ( a b )	match a/(P),b/∅
(P) ( ) Q	apply Involution, Pervasion
( ) a	match a/(P) Q
( )	apply Dominion

Annotated Parens: [(((P)<sub>3</sub> ((P)<sub>5</sub> Q)<sub>4</sub>)<sub>2</sub>)<sub>1</sub> Q]

Ordered Pairs: {(U,1),(U,Q),(1,2),(2,3),(2,4),(3,P),(4,5),(4,Q),(5,P)}

{(U,1),(U,Q),(1,2),(2,3),(2,4),(3,P),(4,5),(4,Q),(5,P)}	initial arrangement
{(U,1),(U,Q),(1,2),(2,3),(2,4),(3,P),(4,3),(4,Q)}	structure sharing
(u,b) (b,c) (c,y <sub>-</sub> )(c,y <sub>-</sub> )	(b,∅) match u/U,b/1,c/2,y <sub>-</sub> /3 4
(u,y <sub>-</sub> ) (u,c) (c,z <sub>-</sub> )(c,y <sub>-</sub> )	match u/U,c/4,y <sub>-</sub> /Q,z <sub>-</sub> /3
(u,y <sub>-</sub> )(u,c) (c,y <sub>-</sub> )(c,z <sub>-</sub> )	match u/2,c/4,y <sub>-</sub> /3,z <sub>-</sub> /Q
{ (U,Q), (U,3),(U,4),(3,P) }	apply Inv.,Pervasion
(u,x <sub>-</sub> ) (u,x <sub>-</sub> )(u,b)	(b,∅) match u/U,b/3,x <sub>-</sub> /P Q
{ (U,4),(3,P) }	(4,∅)} apply Dominion

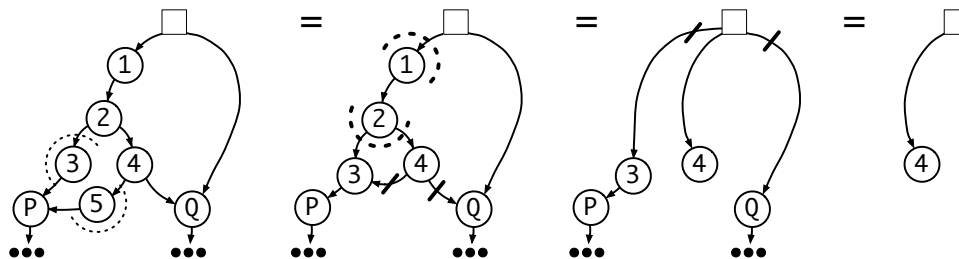
Result: [(∅)<sub>4</sub>]

---

The ordered pair representation is not well suited for parallel pattern-matching. In particular, several pairs are involved in different concurrent matches. When the structure of the problem is expressed as a graph, multiple matching of nodes is easily managed without contradiction or deadlock.

This example also illustrates that the process of reduction can leave some symbolic debris that is not contained by U. In this case it is the pair (3,P). The debris is due to the symbolic modeling decision to represent multiple nesting by separate relational formulas. The Dominion rule matches the container of Variable-P, but not Variable-P itself. This is an example of strict containment: whenever a container is deleted, so is all of its contents. In the symbolic form, when a container is deleted, its symbolic contents remain as inaccessible structure. In the graph form, nodes remain that are disconnected from the universal node of the graph. In the iconic form, the entirety of container and contents are fully deleted, because there are no relational fragments or graph links to support partial deletion.

The parallel graph reduction of modus ponens follows.



Structure sharing that combines Node-3 and Node-5 is shown as a first step (light dotted arcs). Structure sharing is not a reduction rule, it is a technique that condenses arrangements that are identical but for labeling. Next, Node-1 and Node-2 enter into a self-deletion agreement, implementing Involution. At the same time, propagating Pervasion messages disconnect two other nodes. After these reductions, the Dominion rule triggers. Whenever a node has no children, it sends a message to its parent node(s) to delete all other children nodes.

## A.6 Proof of the Subsumption Theorem

The Subsumption Theorem of propositional calculus is transcribed into parens and proved by constructive (identified by +) and destructive (identified by -) application of the LoF computational rules. Capital letters stand in place of arbitrary structure. Again in this example structure sharing is important, this time for the implementation of node construction. When Pervasion is applied constructively, the arrangement (A) must be replicated and relabeled, so that further transformation effects only the replicated form.

---

$((A)(A B))$	initial arrangement
$((A)((A) A B))$	+Pervasion
$((A)(( ) A B))$	-Pervasion

$((A) \quad )$  -Occlusion  
 $A$  -Involution

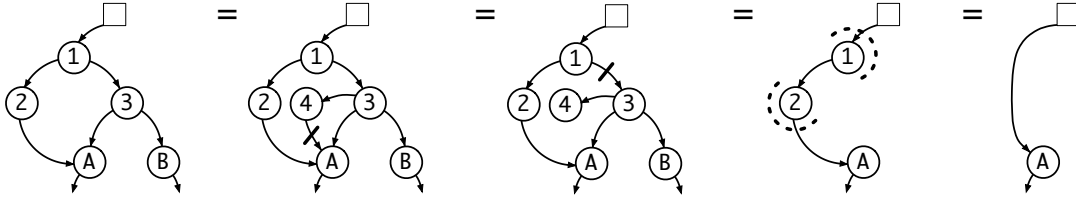
Annotated Parens:  $[(A)_2 (A B)_3]_1$

Ordered Pairs:  $\{(U,1),(1,2),(1,3),(2,A),(3,A),(3,B)\}$

$\{(U,1),(1,2),(1,3),(2,A),(3,A),(3,B)\}$		initial arrangement
$(u,y\_)(u,c) \quad (c,z\_)(c,z\_)$		match $u/1,c/3,y_/2,z_/A B$
$\{(U,1),(1,2),(1,3),(2,A),(3,A),(3,B),(3,4),(4,A)\}$		apply +Pervasion
$(u,y\_)(u,c) \quad (c,y\_)(c,z\_)$		match $u/3,c/4,y_/A,z_/\emptyset$
$\{(U,1),(1,2),(1,3),(2,A),(3,A),(3,B),(3,4), \quad (4,\emptyset) \}$		apply -Pervasion
$(u,b) \quad (b,y\_)(b,y\_)(b,c) \quad (c,\emptyset)$		match $u/1,b/3,c/4,y_/A B$
$\{(U,1),(1,2), \quad (2,A), \quad \}$		apply -Involution
$(u,b) \quad (b,c) \quad (c,y\_)$	$(b,\emptyset)$	match $u/U,b/1,c/2,y_/A$
$\{ \quad (U,A) \quad \}$		apply -Involution

Result:  $[A]$

The graph proof is



The graph proof shows the original arrangement first, followed by the construction of  $(A)_4$ . Here, Node-1 pervades  $(A)_2$ , and also pervades Node-3. Constructive Pervasion permits  $(A)_4$  to be added as a child of Node-3. This triggers a disconnect in the new structure via Pervasion, A and  $(A)_4$  are both children of Node-3. The resulting empty Node-4 triggers Occlusion to delete the entire subgraph of Node-3. Next, Involution deletes Node-1 and Node-2. In the last image, the graph has stabilized with no further reductions.

Each graph variable has an exiting downward link, to emphasize the generality of the transformations. A and B identify arbitrary subgraphs, the predicate calculus equivalent of sets of formulas.

## A.7 A Logic Puzzle

This example is a simple logic puzzle that may typically be found in puzzle books or on exams in a first course on logic. The natural deduction proof requires over a dozen steps, as well as a wise selection of deductive strategy.

The puzzle is described in words and in symbolic logic. It is then transcribed into iconic form and the three void-based transformation rules are applied to arrive at a solution. The same reduction expressed as ordered pairs is also presented.

Here is the puzzle in English:

Premise 1: If he is lying, then  
                     (if we can't find the gun, then he'll get away).  
 Premise 2: If he gets away, then  
                     (if he is drunk or not careful,  
                             then we can find the gun).  
 Premise 3: It is not the case that  
                     (if he has a car, then we can find the gun).  
 Conclusion: It is not the case that  
                     he is both lying and drunk.

Encode the propositions as letters, and encode the logical connectives symbolically:

l = he is lying	d = he is drunk
g = we can find the gun	c = he is careful
a = he will get away	h = he has a car

P1:  $l \rightarrow (\neg g \rightarrow a)$   
 P2:  $a \rightarrow ((d \vee \neg c) \rightarrow g)$   
 P3:  $\neg(h \rightarrow g)$   
 C:  $\neg(l \& d)$

Here is the encoding of the above symbolic logic in parens notation:

P1:	(l) ((g)) a	=	(l) g a	Involution
P2:	(a) (d (c)) g			
P3:	((h) g)			
C:	(((l)(d)))	=	(l)(d)	Involution

The premises are all joined in conjunction, and set to imply the conclusion, using this template:

Deductive Template: (Premise-1 & Premise-2 & Premise-3)  $\rightarrow$  Conclusion

( ((P1)(P2)(P3)) ) C = (P1)(P2)(P3) C      Involution

The problem in iconic form:

( P1 ) ( P2 ) ( P3 ) C

((l) g a) ((a) (d (c)) g) (((h) g)) (l)(d)

Should the iconic structure reduce to ( ), the conclusion is TRUE. Should it reduce to void, the conclusion is FALSE. And should it reduce to a form that contains variables, then the conclusion is contingent on the specific values of the remaining variables. A satisfying binding is easily generated by assigning the remaining variables values that reduce the entire arrangement to ( ).

For display readability, the ordered pairs proof shows only those pairs relevant to a particular transformation. Those pairs not involved in the transformation remain in the descriptive set. Note that in the iconic proof, each rule application step deletes structure. The rule and its application can both be identified simply by observing what structure has been deleted on a given line.

---

((l) g a) ((a) (d (c)) g) (((h) g)) (l)(d)	initial arrangement
((l) g a) ((a) (d (c)) g) (h) g (l)(d)	Involution
( a) ((a) (d (c)) ) (h) g (l)(d)	Pervasion g (l)
( a) ( d (c)) ) (h) g (l)(d)	Pervasion (a)
( a) d (c) (h) g (l)(d)	Involution
( a) d (c) (h) g (l)( )	Pervasion d
( )	Dominion

Annotated Parens: [((l)<sub>6</sub> g a)<sub>1</sub> ((a)<sub>7</sub> (d (c)<sub>9</sub>)<sub>8</sub> g)<sub>2</sub> (((h)<sub>11</sub> g)<sub>10</sub>)<sub>3</sub> (l)<sub>4</sub> (d)<sub>5</sub>]

Ordered Pairs:

{(U,1),(U,2),(U,3),(U,4),(U,5),(1,6),(1,a),(1,g),(6,l),(2,7),(2,8),(2,g),  
(7,a),(8,d),(8,9),(9,c),(3,10),(10,11),(10,g),(11,h),(4,l),(5,d)}

{...(U,3),(3,10),(10,11),(10,g)...} part of initial arrangement  
(u,b) (b, c) ( c,y\_) (c,y\_) (b,∅) match u/U,b/3,c/10,y\_/11 g  
{... (U,11), (U,g)... } apply Involution

{...(U,1),(1,6),(1,a),(1,g),(U,g)...} part of descriptive set  
(u,c) (c,z\_)(c,z\_)(c,y\_)(u,y\_) match u/U,c/1,y\_/g,z\_/a 6  
{...(U,1),(1,6),(1,a), (U,g)...} apply Pervasion g

{...(U,2),(2,7),(2,8),(2,g),(U,g)...} part of descriptive set  
(u,c) (c,z\_)(c,z\_)(c,y\_)(u,y\_) match u/U,c/2,y\_/g,z\_/7 8  
{...(U,2),(2,7),(2,8), (U,g)...} apply Pervasion g

{...(4,l),(1,6),(6,l)...} part of descriptive set  
{...(4,l),(1,4) ...} structure sharing (l)

{...(U,4),(U,1),(1,4),(1,a)...} part of descriptive set  
(u,y\_)(u,c) (c,y\_)(c,z\_) match u/U,c/1,y\_/4,z\_/a

$\{ \dots (U,4), (U,1), \quad (1,a) \dots \}$	apply Pervasion (l)
$\{ \dots (1,a), (2,7), (7,a) \dots \}$	part of descriptive set
$\{ \dots (1,a), (2,1) \quad \dots \}$	structure sharing (a)
$\{ \dots (U,1), (U,2), (2,1), (2,8) \dots \}$	part of descriptive set
$\quad (u,y_-)(u,c) \quad (c,y_-)(c,z_-)$	match $u/U, c/2, y_-/1, z_-/8$
$\{ \dots (U,1), (U,2), \quad (2,8) \dots \}$	apply Pervasion (a)

Remaining Ordered Pairs:

$\{(U,1), (U,2), (U,11), (U,g), (U,4), (U,5), (1,a), (2,8), (8,d), (8,9), (9,c), (11,h), (4,l), (5,d)\}$

$\{ \dots (U,2), (2,8), (8,9), (8,d) \dots \}$	part of descriptive set
$\quad (u,b) \quad (b,c) \quad (c,y_-)(c,y_-) \quad (b,\emptyset)$	match $u/U, b/32, c/8, y_-/9 \quad d$
$\{ \dots \quad (U,9), (U,d) \dots \}$	apply Involution

$\{ \dots (U,d), (U,5), (5,d) \dots \}$	part of descriptive set
$\quad (u,y_-)(u,c) \quad (c,y_-) \quad (c,z_-)$	match $u/U, c/5, y_-/d, z_-/\emptyset$
$\{ \dots (U,1), (U,2), \quad (5,\emptyset) \dots \}$	apply Pervasion d

$\{ \dots (U,1), (U,11), (U,g), (U,4), (U,5), (U,d), (U,9), (5,\emptyset) \dots \}$	
$\quad (u,x_-)(u,x_-) \quad (u,x_-)(u,x_-)(u,b) \quad (u,x_-)(u,x_-)(b,\emptyset)$	match $u/U, b/d, x_-/1 \quad 11 \quad g \quad 4 \quad d \quad 9$
$\{ \dots \quad (U,5) \quad (5,\emptyset) \dots \}$	apply Dominion

$\{(1,a), (9,c), (11,h), (4,l), (U,5), (5,\emptyset)\}$	remaining
$\{ \quad (U,5), (5,\emptyset) \}$	disconnected

Result:  $[( \ )]$

Interpretation: TRUE

The ordered pairs proof must take smaller steps than the iconic proof. For example, Pervasion of  $g$  is recorded as two steps, once for each nested  $g$  in the descriptive set. This is not necessary in the iconic proof because semipermeability of boundaries permits direct access to all nested replications of  $g$ .

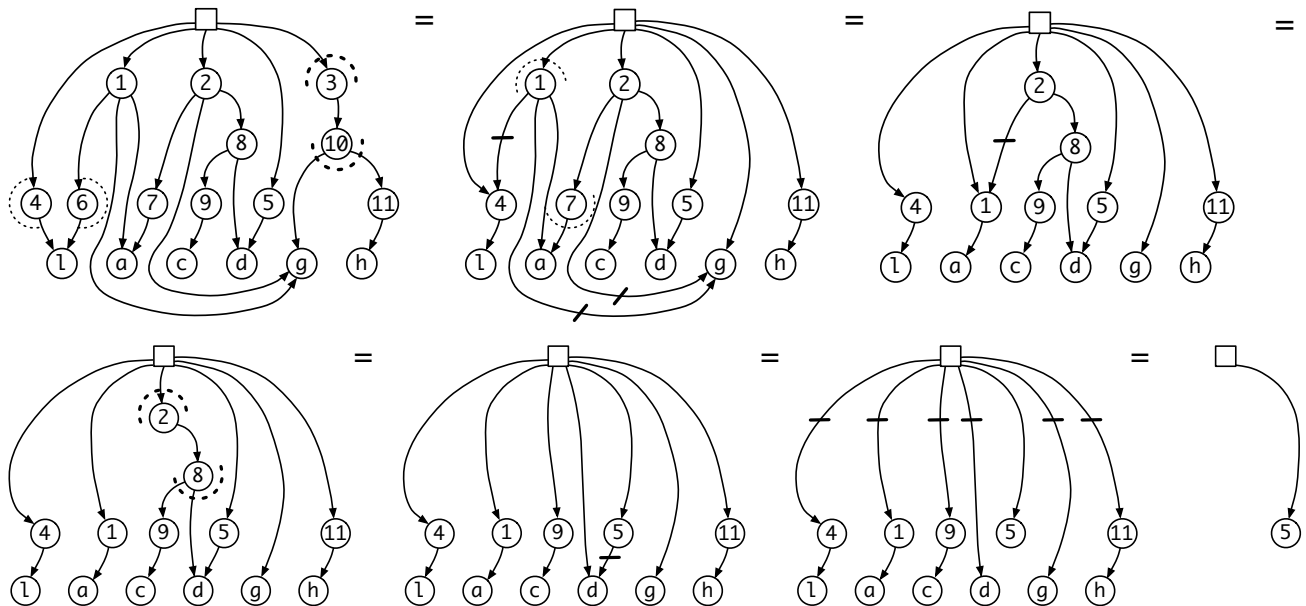
The ordered pairs form also requires structuring sharing. For example, the arrangements  $(l)_4$  and  $(l)_6$  have the same structure and can be condensed into one. This is necessary to prepare the descriptive set for Pervasion of the compound arrangement  $(l)$ .

Finally, the descriptive set contains disconnected pairs after the reduction is completed. These must be deleted via a process of screening for pairs that are not nested within  $U$ .



The additional steps required by the ordered pairs notation (atomic Pervasion steps, structure sharing, and disconnected debris) are artifacts of the ordered pairs notation for containment arrangements. These artifacts themselves are a direct consequence of the symbolic necessity of labeling individual objects rather than treating arrangements spatially. This returns to a central theme of the essay: the strict separation of syntax from semantics and the attendant assurance of formality in symbolic notations imposes several costs. Compared to an iconic formalization, computation is more fragmented and more complicated, representation is artificial and arbitrary, and more powerful visual and spatial tools are unavailable. Symbolic notation does not achieve neutrality with regard to either representation or computation.

The graph reduction of this problem proceeds in six steps,



The initial image shows the deductive problem after each premise has been simplified using Involution. The image indicates the first possible reduction step which is another application of Involution. It also shows an application of structure sharing. The second image incorporates both the rule application and the structure sharing modification, the replicate  $(l)_4$  and  $(l)_6$  arrangements are joined. This image also shows the location of three new applications of Pervasion, and another application of structure sharing. The third image shows these three rule applications and the second application of structure sharing, the replicates  $(a)_1$  and  $(a)_7$  arrangements are joined. This leads to a new application of Pervasion, followed by Involution, and yet another new application of Pervasion. The second to last image shows Dominion triggered by Node-5, reducing the graph to a single empty node. The single node is interpreted as TRUE.

### A.7.1 Cognitive Interpretations

We present outlines of the proof of this puzzle as narratives, using truth tables, natural deduction, and iconic deletion.

#### TRUTH TABLES

We listed 64 possibilities of all combinations of binary values for the six propositions. We then tried each combination for each binary connective in the Premises and Conclusion, and for each connective when the Premises and Conclusion were combined. This required fifteen tests over each of the 64 possibilities. The final test showed all possibilities were TRUE, thus solving the puzzle.

To recapitulate: We computed nearly one thousand possible binary situations by referring to the definition of each connective in a table of possibilities. The job was clerical and did not require much thought.

#### NATURAL DEDUCTION

We need to show  $\neg(l \ \& \ d)$ . By adopting a contradiction strategy, we can assume  $(l \ \& \ d)$ , which in turn provides both  $l$  and  $d$  as premises by AND-simplification. We then try to anchor the truth values of the other propositions.  $l$  provides  $(\neg g \rightarrow a)$  by modus ponens on Premise-1. We can get  $\neg g$  by rewriting Premise-3 using conditional exchange, then applying double negation and DeMorgan to get the conjunction  $(h \ \& \ \neg g)$ , and then applying AND-simplification to give both  $h$  and  $\neg g$ .  $\neg g$  in turn provides  $a$  by modus ponens on what remains of Premise-1.  $a$  provides  $(d \vee \neg c) \rightarrow g$  by modus ponens on Premise-2.  $(d \vee \neg c) \rightarrow g$  converts to  $d \rightarrow g$  by conditional disjunction followed by AND-simplification. We have  $d$ , so that gives  $g$  by modus ponens. Finally having both  $g$  and  $\neg g$  provides the needed contradiction, permitting the conclusion  $\neg(l \ \& \ d)$ .

To recapitulate: We developed a contradiction strategy to find the value of propositions, followed implications to get the values of other propositions, engaged in several different transformations to expose more propositions, and finally ended up with two contradictory values which then solves the puzzle. The proof took fourteen steps, and required strategic thinking, knowledge of a dozen transformation rules, and both planning and exploring.

#### ICONIC DELETION

We applied Involution to simplify two of the premises. We then put each premise into a container, and put them all into U, together with the conclusion. Premise-3 now has a double-boundary to delete using Involution. With g and (l) in U, replicates of each can be deleted from deeper nestings using Pervasion. This leaves (a) as Premise-1, which in turn permits the deletion of (a) from Premise-2. What remains of Premise-2 is now inside a double-container. Deleting that exposes both d and (c) in U. We also have (d) in U, so the nested d within (d) can be deleted. We're left with ( ) and a bunch of other stuff that is deleted via Dominion.

To recapitulate: We put everything into one container, deleted irrelevant structures, deleted more irrelevant structures that became exposed, and solved the puzzle when nothing but the empty container was left. The job was clerical, took seven steps, and did not require much thought.

## A.8 Use of Virtual Arrangements

This final example shows the application of virtual arrangements to Boolean minimization. This type of problem is common in the design of silicon circuits. Virtual arrangements are void-equivalent forms that are postulated to exist as queries. If useful, they are actualized as symbolic notation; if not useful, they are ignored.

The purpose of this example is to show that relatively difficult, pragmatic Boolean algebra problems (of which the example is but a small toy problem) can be addressed using iconic representation and reduction techniques.

Only the parens minimization is shown. The ordered pairs reduction is bulky. The initial problem, for example, consists of 39 containment pairs. However the primary difficulty with the symbolic notation is that the void-equivalence technique of virtual arrangements relies upon Deep Pervasion, which is rather clumsily expressed by paths of nestings through shallow Contains relations.

The problem is to reduce a small Conjunctive Normal Form expression to the minimum number of occurrences of variable letters. The initial six term expression contains 24 variable mentions. Expressed in the notation of Boolean algebra, the initial problem would be written as

$$1'23'4' + 1'23'4 + 12'34' + 1234' + 12'34 + 1234$$

This would transcribe into logic as

$$\begin{aligned} &(\text{NOT } 1 \text{ AND } 2 \text{ AND NOT } 3 \text{ AND NOT } 4) \text{ OR } (\text{NOT } 1 \text{ AND } 2 \text{ AND NOT } 3 \text{ AND } 4) \text{ OR} \\ &(1 \text{ AND NOT } 2 \text{ AND } 3 \text{ AND NOT } 4) \text{ OR } (1 \text{ AND } 2 \text{ AND } 3 \text{ AND NOT } 4) \text{ OR} \\ &(1 \text{ AND NOT } 2 \text{ AND } 3 \text{ AND } 4) \text{ OR } (1 \text{ AND } 2 \text{ AND } 3 \text{ AND } 4) \end{aligned}$$

The Boolean expression is first transcribed into parens using Table 10.1, and then Involution is applied, in this case nine times. For convenience, each resulting parens arrangement is labeled with a capital letter.

[ (((1))(2)((3))((4))) (((1))(2)((3))(4)) ((1)((2))(3)((4)))  
 ((1)(2)(3)((4))) ((1)((2))(3)(4)) ((1)(2)(3)(4)) ]

Apply Involution,

A B C D E F  
 [(1 (2) 3 4) (1 (2) 3 (4)) ((1) 2 (3) 4) ((1)(2)(3) 4) ((1) 2 (3)(4)) ((1)(2)(3)(4))]

The minimization consists only of insertion of virtual arrangements into actual arrangements. Each virtual arrangement is reduced by the pervasion of its local context. Successful reductions result in deletion of parts of the actual arrangement. Unsuccessful reductions result in no change. We show one unsuccessful insertion first, but then show only those that are successful.

UNSUCCESSFUL EXAMPLE, insert A into the (2) arrangement in B,

(1 (2) 3 4) (1 (2	) 3 (4))	sub-problem A into B
(1 (2) 3 4) (1 (2 ^ (1 (2) 3 4) ^)	3 (4))	+Pervasion
(1 (2) 3 4) (1 (2 ^ ( ( ) 4) ^)	3 (4))	-Pervasion
(1 (2) 3 4) (1 (2 ^	) 3 (4))	-Occlusion
(1 (2) 3 4) (1 (2	) 3 (4))	no change

## SUCCESSFUL REDUCTIONS

Several reductions are shown separately. The following steps reduce arrangements A and B to a single simpler arrangement.

(1 (2) 3 4) (1 (2) 3 (4	))	sub-problem A into B
(1 (2) 3 4) (1 (2) 3 (4 ^ (1 (2) 3 4) ^)	)	+Pervasion
(1 (2) 3 4) (1 (2) 3 (4 ^ (	) ^)	-Pervasion
(1 (2) 3 4) (1 (2) 3	)	-Occlusion
(1 (2) 3 4	) (1 (2) 3)	sub-problem B into A
(1 (2) 3 4 ^ (1 (2) 3) ^)	(1 (2) 3)	+Pervasion
(1 (2) 3 4 ^ (	) ^)	-Pervasion
(1 (2) 3)		-Occlusion

This last sequence of four reductions can be expressed as a pattern-variable theorem, Subsumption,

## SUBSUMPTION

$$(A) (A B) = (A)$$

Iconic equations such as this are more powerful and more general than conventional symbolic equations because of the pattern-variables (Section 6). The two pattern-variables in the arrangement (A B) partition the objects within the common container into all possible collections. Thus, Pattern-variable-A in (A) matches any subset of objects in (A B).

Next arrangements C and D are reduced to a simpler arrangement by the same steps.

((1) 2 (3) 4) ((1)(2	) (3) 4)	sub-problem C into D
((1) 2 (3) 4) ((1)(2 ^((1) 2 (3) 4)^)	(3) 4)	+Pervasion
((1) 2 (3) 4) ((1)(2 ^((	) ^)(3) 4)	-Pervasion
((1) 2 (3) 4) ((1)	(3) 4)	-Occlusion
((1)	(3) 4)	-Subsumption

Again the entire sequence could have been expressed by a single theorem, Cancellation,

$$(A B) (A (B)) = (A)$$

Four arrangements now remain,

B	D	E	F
[(1 (2) 3)	((1)(3) 4)	((1) 2 (3)(4))	((1)(2)(3)(4))]

The insertion of virtual arrangements continues, D into E, and D into F.

((1)(3) 4) ((1) 2 (3)(4	))	sub-problem D into E
((1)(3) 4) ((1) 2 (3)(4 ^((1)(3) 4)^)	)	+Pervasion
((1)(3) 4) ((1) 2 (3)(4 ^((	) ^))	-Pervasion
((1)(3) 4) ((1) 2 (3)	)	-Occlusion
((1)(3) 4) ((1)(2)(3)(4	))	sub-problem D into F
((1)(3) 4) ((1)(2)(3)(4 ^((1)(3) 4)^)	)	+Pervasion
((1)(3) 4) ((1)(2)(3)(4 ^((	) ^))	-Pervasion
((1)(3) 4) ((1)(2)(3)	)	-Occlusion

The remaining arrangements are

B	D	E	F
[(1 (2) 3)	((1)(3) 4)	((1) 2 (3))	((1)(2)(3))]

The virtual insertion process continues with E into F. This application of Pervasion reduces the entire remaining arrangement

(1 (2) 3) ((1)(3) 4) ((1) 2 (3)) ((1) (2	) (3))	remaining
(1 (2) 3) ((1)(3) 4) ((1) 2 (3)) ((1) (2 ^((1) 2 (3)) ^)	(3))	+Pervasion
(1 (2) 3) ((1)(3) 4) ((1) 2 (3)) ((1) (2 ^ (	) ^) (3))	-Pervasion
(1 (2) 3) ((1)(3) 4) ((1) 2 (3)) ((1)	(3))	-Occlusion
(1 (2) 3)	((1) (3))	-Subsumption

Minimization result: [(1 (2) 3) ((1)(3))]

Transcription into Boolean algebra:  $1'23' + 13$

Obviously this process is not human friendly. However, the simplicity and repetition of the algorithm make it an excellent candidate for automation. An additional advantage is that the size of the problem never increases, so that no additional memory for storage of intermediate results is needed.