WHAT'S THE DIFFERENCE? Contrasting Boundary and Boolean Algebras

William Bricken

October 2005

ABSTRACT: There is a common misconception that boundary algebra is isomorphic with two-element Boolean algebra. The paper describes a dozen deep structural differences that are incompatible with isomorphism. Boundary algebra does not support functional morphisms of any kind because it does not support functions. It does support a partial order relation. A one-to-many mapping between boundary and Boolean algebras emphasizes the difference between the two systems. This mapping shows that boundary algebra subsumes Boolean algebra within a formally smaller structure. Boundary algebra applies to n-element Boolean algebras, and does not have group theoretic structure.

CONTENTS

1 IS BOUNDARY BOOLEAN?

The Laws of Form (LoF) [Spencer-Brown(69)] has been identified as "a provocative and economical notation for two-element Boolean algebra" [Wikipedia topic: Laws of Form]. This is entirely correct, however LoF is not a Boolean algebra, it is a fundamentally different, formally smaller system. LoF can be read as Boolean algebra through a one-to-many mapping, one form in LoF covers many forms in Boolean algebra and in propositional logic. The economy of syntax comes from a deeper source, an economy of semantics. Provocatively, a completely different mathematical system is needed to remove the functionally redundant structure and meaning within logic itself.

An antiquated but still unresolved mathematical claim is that LoF is just another notation for Boolean algebra, that the two systems are isomorphic [Orchard(1975), Banashuski(77), Cull and Frank(79), Kohout and Pinkava(80), Schwartz(81), Meguire(2003)]. Typical of this perspective is that LoF is "... simply another axiomatization of Boolean algebra" [Gould(77)].

Isomorphism means that the structures of LoF and of Boolean algebra are essentially the same, only the labels have been changed. It invalidates any claim that LoF is mathematically interesting, and constrains the innovations within LoF to the trivial. Another mathematically informed viewpoint exists, that LoF is "... not an arbitrary new calculus, but that particular calculus which can let us see deeper into the nature of mathematics" [Whyte(72)].

In the sequel, "simple logic", "propositional calculus", and "Boolean algebra" are used interchangeably, in recognition of the isomorphism between the three. In context, Boolean algebra also refers to the n-element algebra of subsets. "LoF" and "boundary algebra" are also used interchangeably, the former in recognition of Spencer-Brown's book, Laws of Form, that introduced some of the new mathematical concepts, and the latter as a somewhat appropriate name for the mathematical system. The first publication of boundary forms used for logic was C. S. Peirce's "existential graphs" at the turn of the twentieth century [Peirce (33), Roberts(73), Kauffman(2001)].

1.1 The Differentiator

A differentiator between boundary and Boolean algebra is needed, a clear difference that raises LoF to intrinsic value, an irrefutable identification of mathematical structure that is present in one system and absent in the other.

Without a differentiator, boundary algebra does not improve upon conventional Boolean techniques, it is of interest perhaps only as a new data structure for computation or as a unique metaphor for some cognitive perspectives. Isomorphic systems can have notational but not conceptual novelty; the differentiator must be semantic rather than syntactic.

Should the differentiator show that boundary algebra has more expressive power than simple logic, then LoF is an extension, an elaboration that may or may not be useful. However, boundary algebra is equally as expressive as logic, while at the same time having intrinsically less structure than logic. That is, logic contains superfluous structure, the same can be done with less. The embedded redundancy within logic is well known, and is the source of the plethora of transformation rules within logical systems.

1.2 The Message

The main message in this paper is that boundary algebra differs from Boolean algebra in fundamental semantic ways, offering new perspectives on many mathematical conventions and systems. Boundary algebra excels at Occam's razor, it is mathematically more elegant than Boolean systems. Should logic be considered as a mathematical foundation, boundary algebra is the foundation upon which that foundation is built.

Boundary algebra abandons most of the concepts of conventional algebra. Cardinality, associativity, commutativity, arity and functions are not within LoF; imposing any one of them upon boundary algebra changes it into something else. Imposing group theoretic concepts on boundary algebra converts it into a Boolean algebra, while eliminating the efficiency and beauty of LoF's spatial mathematics.

Section 2 presents a variety of new boundary algebra concepts and techniques that can enrich the mathematical enterprise.

Section 3 identifies many differentiators -- semantic void/single constant, inside and outside of forms, non-linear notation, object/operator unification, semipermeable boundaries/operational transparency, void-equivalence -- and nominates a prime differentiator that is both simple and incisive:

> Boundary algebra has a one-to-many map to Boolean algebra.

Section 4 shows that boundary algebra is not restricted to two-element Boolean algebras, it applies to any finite Boolean algebra.

Section 5 demonstrates that the structures within boundary algebra do not have the group theoretic structure of an algebraic group.

In summary, Section 6 lists the essential differences between boundary and Boolean algebras.

1.3 Boolean Algebra

An algebraic system is a set of elements together with one or more operations on that set. Arithmetic systems are based on a set of constants only, while algebraic systems add variables.

Systems usually incorporate specified axioms or rules for reducing expressions. These rules can contain variables that stand in place of arbitrary expressions. When the rules are formulated using equality relations that place constraints on the possible values of a variable, the algebra is universal. An algebraic group, for example, is a specific type of system that meets group theoretic constraints of closure, associativity, identity and inverse.

An equation is a mathematical assertion that two expressions are equivalent in value, but not in form. Expressions with the same form are identical; expressions with different forms but the same value belong to the same equivalence class. Systems that are formulated using equations incorporate the powerful and familiar computational techniques of substitution of equals for equals and global replacement of forms. Equations also provide the familiar proof techniques of transforming one side of the equation into the other, and transforming both sides into the same form.

A Boolean algebra consists of a set, S, together with two binary functions, generically called meet and join, and one unary operation, the complement.

Boolean algebra is commonly characterized by ten constraint rules, not all of which are independent. These rules include associativity, commutativity, distributivity, complement and zero/unit element constraints for each of the binary operators.

Propositional logic has two elements, called truth values (TRUE, FALSE). Boolean algebra, however, is not limited to two elements. The representation theorem of finite Boolean algebras states that there is an isomorphism between any Boolean algebra and the algebra of subsets. A set can have any number of elements. The algebra of subsets is the canonical, so to speak, Boolean algebra of n elements. Propositional logic is a special case of the algebra of subsets with two elements.

2 BOUNDARY SYSTEMS

Boundary algebra is not a conventional algebra, although it does incorporate equations and variables. As will be shown, boundary algebra lacks binary and unary functions. "Boundary" refers to a unique type of element, one that has spatial extent. Boundaries are enclosures that have an inside and an outside. But boundaries are not functions, they are relations between the inside and the outside of the boundary. Boundary algebra, therefore, does not support many of characteristics of conventional algebras.

When parentheses are used typographically to represent boundaries, the forms of boundary algebra consist of all possible well-formed parentheses configurations. For example, some of these forms are:

( ) (( )( )( )) ((( ))) (((( )))) ((( ))(( )(( ))))

Other representations of the domain of boundary algebra include non-overlapping stacks of blocks, shapes of branching trees, node-and-link networks, topological maps, and branching paths that can cross and re-cross a closed loop. These representations follow formal, mathematical transformation rules, even though they are very different than the words and tokens of conventional algebras.

Surprising interpretations of boundary algebra include simple logic (propositional calculus), Boolean algebra, and combinational computer circuits. Since logic is so central to mathematics, to computation, and to rationality, a new formal method that redefines Boolean concepts is of intrinsic interest.

2.1 A Boundary Arithmetic

In the book Laws of Form, G. Spencer-Brown presents an equational system that uses spatial enclosures as objects of computation. Spencer-Brown's arithmetic is particularly succinct because there is only one constant, the enclosure or boundary ( ).

The boundary arithmetic presented in Laws of Form consists of two initial equations relating replications of the one constant.

( ) ( ) = ( ) CALLING

(( )) = CROSSING

The constant, called Mark, is a two-dimensional enclosure, ( ), that distinguishes the inside of a closed loop.

The left-hand-sides of Spencer-Brown's equations are quite natural, they identify the two configurations that are possible when a replicated Mark is drawn on the same page as the original Mark. The replicate can be drawn outside of the original, in which case the two Marks are sharing space. The replicate can be drawn on the inside of the original, in which case the two Marks are nested, the inner Mark is bounded by the outer Mark.

Although the two configurations of replicates are structurally determined, the value of each configuration on the right-hand-side is a choice, defining a particular boundary arithmetic. In Spencer-Brown's system, two Marks sharing space are equivalent to one Mark. Two nested Marks are equivalent to no Marks. Thus the equations define two equivalence classes, those that reduce to Mark and those that reduce to nothing at all.

Different initial equations generate different types of boundary arithmetic. For example, the rule of CALLING can be omitted, leaving no mechanism to remove replicates sharing the same space. Cardinality is thus introduced, since a specific number of identical forms can occupy a space. The situation is analogous to standard and modular arithmetic; the number of constants and the rules of addition differ in each. The standard arithmetic of numbers does not have an upper limit to cardinality, while modular arithmetic sets a specific upper limit to the number of constants in the system. Boundary arithmetic has a modulus of one.

2.1.1 Boundary Operations

CALLING and CROSSING identify two operations, SHARING and BOUNDING, that can be illustrated abstractly as:

2.1.1.1 Sharing Space

SHARING is not a binary function, it is more like set membership. Forms simply share the same unstructured space. A space can accommodate any number of forms, a boundary can enclose any number of forms. There is no ordering or grouping of forms in space, since the void-based space does not have properties.

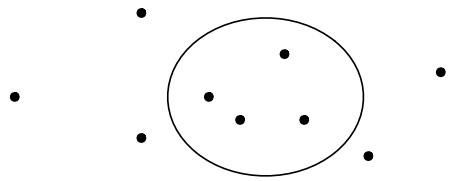
A rule of associativity is necessary when a function or relation has a specific arity, or number of arguments. Associativity of SHARING operations is a meaningless concept, since the space of SHARING supports no grouping properties and the operation of SHARING applies to any number of contents in a space, including none.

A rule of commutativity is necessary when a function or relation has a specific metric, or location of arguments. Commutativity of SHARING operations is a meaningless concept, since the space of SHARING supports no ordering or location properties.

2.1.1.2 Bounding Space

BOUNDING serves to contain forms within a closed boundary. There is space inside a boundary, the same kind of space that is outside the boundary. A boundary too is insensitive to the number and structure of forms inside it. Boundaries do support a type of grouping, they group by containment the forms that are their contents.

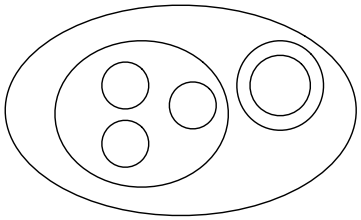
Boundaries distinguish inside from outside. Since there can be any number of forms on either side, BOUNDING does not have an arity. Pictorially, the abstract dots below represent arbitrary configurations, their relative positions are defined only by the boundary.



Boundaries can contain other boundaries, forming a nested structure. The nesting of boundaries is a semantic inclusion relation that is overtly visible within the syntax of the system. Thus, the parenthesis structure

((( )) (( )( )( )))

implies only a collection of nested boundaries, and should be read as a nested spatial structure:



2.1.2 Nothing

Boundary systems permit semantic use of the absence of a Mark, called Void. The rule of CROSSING has an unusual right-hand-side; the left-hand configuration is equal to no configuration at all, it is Void. CROSSING is applied to the nested form, (( )), by deleting or erasing it. Technically CROSSING permits void-substitution, nothing is substituted for something.

Void has no representation, but it does have a natural indicator: Void is the inside of an empty Mark. More fundamentally, Void is the substrate of all Marks, it is the blankness of the page underlying every form. Void pervades all forms, it is both inside and outside of every Mark. As a consequence, Void is unique and singular, there is only one.

2.1.3 Void-Substitution

Each boundary arithmetic equation identifies an available void-substitution; the right-hand-side is equal to the left-hand-side, but with some structure deleted. The deleted structure is equivalent to no structure at all. Computation proceeds purely by void-substitution (i.e. by deletion or erasure); there are no concepts such as rearrangement of forms, look-up tables to determine values, binary functions that transform forms, right- and left-sided functions, cardinality, or truth values.

Computational steps are determined by parallel pattern-matching or by parallel network reduction. The sequence of rule application is irrelevant; convergence is assured by

THE PRINCIPLE OF VOID-EQUIVALENCE

Void-equivalent structure is syntactically irrelevant and semantically inert.

Thus, boundary representation is unconventional is several ways: a semantic void; a single constant with an outside and an inside; a unstructured representational space that can be shared by any number of forms; and a non-linear, non-sequential notation.

2.2 A Boundary Algebra

The two equations of boundary arithmetic generalize quite naturally to accommodate variables, and thus to construct an algebra. One convenient set of initials for boundary algebra consists of three equations, each identifying void-equivalent structure.

```
(( ) A)  =                    OCCLUSION

 ((A))   =  A                 INVOLUTION

A {A B}  =  A {B}             PERVASION
```

Capital letters refer to arbitrary boundary forms, which may include zero, one or more subforms. OCCLUSION and INVOLUTION are generalizations of CROSSING, while PERVASION generalizes CALLING to arbitrary configurations and to arbitrary depths of nesting.

Each equation specifies transformations that do not change the value of a boundary form. This specification is a choice. Different initial equations lead to different types of boundary algebra which support different interpretations.

Each equation identifies void-equivalent structure. The equations can be applied in either direction, left-to-right using void-substitution to delete structure, or right-to-left constructively to add void-equivalent structure. OCCLUSION serves as a halting condition, INVOLUTION deletes paired boundaries, and PERVASION serves to delete forms that have the same pattern.

2.2.1 Pervasion

PERVASION is a powerful reduction technique with no analogs in conventional mathematics. The curly braces are meta-symbols indicating any number of intervening boundaries, including none. PERVASION applies regardless of depth of nesting, generalizing the fundamental idea of the pervasiveness of Void.

The rule of PERVASION renders boundaries semipermeable. What is outside is arbitrarily inside, while what is only inside stays inside. Since matching forms can be deleted from or added to deeper nestings, inward boundaries are transparent to replicates.

In the following example of boundary algebra computation, each line is an application of one of the void-substitution rules:

```
( ((A)) ((A)(B)(C)) )          boundary form
(   A   ((A)(B)(C)) )          involution
(   A   (( )(B)(C)) )          pervasion
(   A               )          occlusion
```

The computation by deletion maintains equivalence; therefore,

```
( ((A)) ((A)(B)(C)) ) = (A)
```

2.2.2 Virtual Forms

Void-equivalent forms are everywhere that Void is, which is everywhere throughout a configuration. For example, consider this demonstration:

((a) a b)	boundary form
(( ) a b)	pervasion
	occlusion

That is, ((a) a b) = .

Since this form is void-equivalent, and thus inert, replicates can be placed constructively anywhere in any space. Here, added replicates are underlined:

= ((a) a b) = ((a) a b) ((a) a b) = ((a ((a) a b)) a b)

The approach of conventional token-based mathematics is to represent forms explicitly, but this would be quite awkward for void-based boundary mathematics. Instead, void-equivalent forms are present implicitly. They can be raised to awareness, used as patterns, and then discarded without effecting the value of a computation. Void-equivalent forms are syntactically virtual.

2.2.3 The Robbins Problem as an Example

The Robbins Problem concerns the axiomatic basis of Boolean algebra. Huntington developed a set of axioms for Boolean algebra that included Associativity, Commutativity, and one equation [Huntington(33)]. Robbins asked: if the one equation were slightly different, does it still constitute a Boolean algebra? This seemingly difficult problem achieved recognition when an automated theorem prover [McCune(96)] used over 17000 steps to answer in the affirmative.

Huntington's and Robbin's equations are presented below as Boolean and boundary forms:

	Boolean	boundary
Huntington	(a' ◊ b)' + (a' ◊ b')' = a	((a) b) ((a)(b)) = a
Robbins	((a ◊ b)' + (a ◊ b')')' = a	((a b) (a (b))) = a

◊ is an undefined binary Boolean operator. Robbins question is interesting due to the apparent symmetries between the two equations. The boundary patterns highlight the essence of the problem: the Huntington "a" is complemented on the

left side of the equation but not on the right, while the Robbins "a" is not complemented. Instead the complement boundary has been factored out to apply to the entire expression.

Using virtual forms, the boundary algebra proofs of both equations are quite simple. The objective here is not to establish the smallest set of axioms, but to show that the Huntington and the Robbins axioms are equivalent in complexity when the three rules of boundary algebra are assumed as a basis.

The boundary forms do not have structural concepts of Associativity and Commutativity; grouping and ordering are not relevant to the boundary proof.

Huntington ((a) b) ((a)(b))

```
((a) b        ) ((a)(b)          )          boundary form
((a) b        ) ((a)(b) ^((a) b)^)          virtual per
((a) b        ) ((a)(b) ^(    b)^)          per
((a) b        ) ((a)    ^(    b)^)          per
((a) b        ) ((a)              )          forget virtual
((a) b ^((a))^) ((a)              )          virtual per
((a) b ^(   )^) ((a)              )          per
                ((a)              )          occ
                  a                          inv
```

Robbins ((a b) (a (b)))

```
((a   b        ) ( a (b)          ))         boundary form
((a   b        ) ( a (b) ^( a  b)^))         virtual per
((a   b        ) ( a (b) ^(    b)^))         per
((a   b        ) ( a     ^(    b)^))         per
((a   b        ) ( a               ))         forget virtual
((a   b ^( a )^) ( a               ))         virtual per
((a   b ^(   )^) ( a               ))         per
(                ( a               ))         occ
                   a                         inv
```

The proofs of each equation are almost identical. These proofs do not directly address the question as to whether or not the Robbins equation is sufficient to establish a Boolean algebra, because the transformations occur in a boundary algebra system that is simpler than (underneath) Boolean algebra. Since boundary algebra has Boolean algebra as an interpretation, the proofs do establish that there is no essential difference between the two equations.

"The Robbins problem has the flavor of a fairy tale that shows that finally there is an escape from the artificial labyrinth made by insisting on existence rather than allowing non-existence." [Kauffman(90)]

Thus, boundary computation is also unconventional is several ways: void-substitution rather than rearrangement; absence of cardinality, commutativity, associativity, arity, and functions; structural parallelism; pervasive operations; and use of virtual forms.

2.3 Some Interpretations

An interpretation is a mapping between systems, usually involving a specific set of rules or constraints. Boundary algebra has disparate rules sets and interpretations. Here, interpretations for elementary logic, for positive integers, for three-valued logic, and for n-element Boolean algebra are mentioned as examples.

2.3.1 Two-element Boolean Algebra

One of the interpretations of boundary algebra is propositional logic, which is a two-element Boolean algebra. The interpretation as two-element Boolean algebra arises quite naturally from the initial rules of CROSSING and CALLING, and follows Spencer-Brown's presentation of LoF.

The map between the two systems can be expressed succinctly:

Boundary	Boolean	Logical
( )	1	TRUE
(a)	~a	NOT a
(a) b	~a v b	a IMPLIES b

The boundary is interpreted as the truth value TRUE and as the unary complement, NOT, and as the binary connective, IMPLIES.

The same transformations, reductions, evaluations and proofs that can be conducted in conventional logic can be conducted in boundary algebra [Bricken and Gullichsen(89)]. The reverse is not true.

The set of well-formed enclosures combined with the reduction rules CROSSING and CALLING is sufficient to fully characterize truth functional evaluation in logic, and the abstract signal propagation behavior of combinational digital circuits. The set of enclosures containing variables (which represent arbitrary forms) combined with the rules OCCLUSION, INVOLUTION and PERVASION is sufficient to fully characterize propositional logic and Boolean algebra.

2.3.2 Integers

Boundary forms can be interpreted for integers as well as for logic and for Boolean algebra [Kauffman(85), Bricken(92), James(93), Kauffman(95)]. The interpretation for integers shows that the Spencer-Brown initials for the arithmetic are limited to a particular application (logic). Although the principles of computing with boundaries remains the same, changing the initial boundary equations can result in a different, equally useful system.

For integers, the boundary can be interpreted both as a unit integer and as a doubling function, using the following mapping:

Boundary	Integers
	0
( ) = *	1
(x)	2x
x y	x + y

0 is Void. To emphasize the atomic structure of the unit integer, the empty Mark is collapsed to a token, *, without an inside. SHARING is addition, while BOUNDING is doubling (multiplication by 2).

For the interpretation as integers, CALLING and CROSSING are not valid rules. Instead, the single arithmetic rule is:

( ) ( ) = (( )) DOUBLE

This boundary algebra of integers includes two initial equations. The algebraic POWER rule generalizes the DOUBLE rule of the arithmetic.

x x = (x) POWER

(x) (y) = (x y) DISTRIBUTION

2.3.3 Three-valued Logic

LoF has been extended to include contradictory (imaginary) logic values by adding a second constant, $(1)_1$, the Reentrant Mark [Kauffman(85), Kauffman (87), Shoup(93), Hellerstein(97)]. The subscripted parenthesis notation here indicates that the Reentrant Mark and its contents, labeled subscript-1, reenters into the space of its contents, labeled 1.

Reentry can occur any number of times. The beginning sequence of equivalent forms generated by a Reentrant Mark is:

$(1)_1 = ((1)_1) = (((1)_1)) = ((((1)_1))) = ...$ REENTRANT MARK

Reentrant forms can be interpreted as infinite structures. Reentrant operations can be interpreted as an infinite sequence of operations that result in a fixed-point value.

In the case of the Reentrant Mark entering its own space, the fixed-point value does not stabilize. However infinite reentry is a natural interpretation of all boundary algebra forms, usually without undermining the value of the form. Both of Spencer-Brown's initials for the arithmetic, for example, can be viewed as infinite reentrant forms:

$= ((1))_1 = ((((1))_1)) = (((((1))_1))) = ...$ REENTRANT CROSS

$( ) = 1 ( )_1 = ( ) 1 ( )_1 = ( ) ( ) 1 ( )_1 = ...$ REENTRANT CALL

One set of initial equations for the reentrant boundary algebra [Varela(75), Varela(79)] applies to the two constants { ( ), $(1)_1$ }:

$( ) A = ( )$ DOMINANCE

$(( )) =$ ORDER

$((1)_1) = (1)_1$ CONSTANCY

$(1)_1 = (1)_1 (1)_1$ NUMBER

2.3.4 N-element Boolean Algebra

Section 4 focuses on the interpretation of boundary algebra as finite Boolean algebra with any number of elements.

For a two-element Boolean algebra, the Mark is the entire Universe; the set upon which the system is based contains Mark as the only element. For an n-element Boolean algebra, the Mark is still the entire Universe, however the set upon which the system is based contains several (n) elements. A six-element Boolean algebra, for example, would have the following definition for Mark:

$( ) = a\ b\ c\ d\ e\ f$

Boolean algebra and set theory provide definitions of operations in the form of equations, however they do not provide algorithms to compute the result of an operation, say for example, the intersection of two sets. In contrast, the three void-based rules of boundary algebra provide both structural definitions and algorithms for computing the results of set operations.

3 A HOST OF DIFFERENTIATORS

The previous sections present several fundamental differences between conventional mathematical techniques and those of spatial boundary algebra. There are many candidates for a primary differentiator, since boundary and Boolean algebras have little in common. Unexpectedly, the novel conceptual structure of LoF is sufficient to subsume both propositional logic and Boolean algebra.

Many published technical descriptions of LoF begin with the conventional and then unfortunately ignore or misinterpret the structure of LoF itself. This exposition has avoided mapping between boundary and Boolean systems in order to emphasize that boundary algebra has unique concepts and techniques of its own. This approach preserves what may be the most interesting aspects of boundary algebra:

1) Logic and human rationality can be cast within a formal system that is fundamentally different than the one that has evolved over the last two millennia.

2) Digital computation and semiconductor design can be cast within a formal system that is fundamentally different than the binary zeros and ones that have defined computer functioning since its inception.

The primary differentiator, that there is a one-to-many mapping between Boolean and boundary domains, is described next, followed by discussion of several other potential candidates that also differentiate boundary and Boolean algebras.

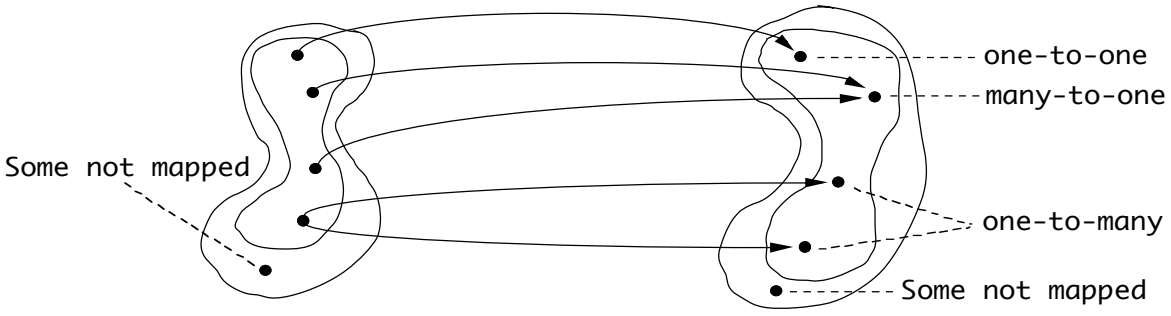
3.1 Mapping

Conventional mathematics addresses three common kinds of relations: equivalence, order, and function. Equivalence classes have been presented in the form of equations. A primary definition of the order relation is that of containment. Boundary forms are clearly ordering relations. Functions map between two sets, the domain and the range, and are limited by two constraints:

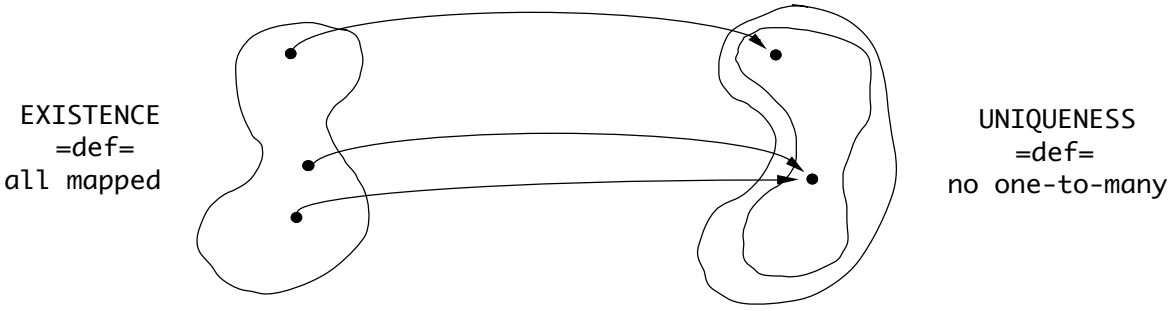
Existence: All the elements of the domain are mapped to the range.
Uniqueness: Every domain element maps to only one range element.

A one-to-many relation cannot be functional, since uniqueness is violated. So there is no functional mapping between boundary and Boolean systems. Only a

relation can map boundary to Boolean systems; this is the order relation, discussed later in this Section.



Relational Mapping



Functional Mapping

3.1.1 One-to-Many

There is a one-to-many map between the structures of boundary and Boolean algebra. A one-to-many mapping asserts that the One system is essentially smaller and consequently more efficient that the Many system. Consider this portion of the one-to-many map:

Boundary	Boolean	Logical
( )	1	TRUE
( )	~0	NOT FALSE
( )	1 v 0	TRUE OR FALSE
( )	~0 v 0	FALSE IMPLIES FALSE
( )	~(0 v 0)	NOT (FALSE OR FALSE)

The same Mark maps onto syntactically different Boolean and logical forms. For example, the Boolean algebra concepts of 1, ~0, and 1 v 0 each express different ideas with different notation. These forms reduce to the same value, however the reduction involves taking computational steps (i.e. making substitutions) to change one form into the other. In the boundary system, no step can be taken, since there is no difference in the representation of the three.

The one-to-many correspondence between forms in each system provides an interpretative freedom. The Mark can be freely read as a constant, as a unary complement, as a binary function between inside and outside, as a function operating on arguments on its inside, or as the representation of a (infinite) number of other Boolean forms. Thus, the one-to-many map transcends syntactic mapping and interpretation, it provides choice at the level of reading the form.

3.1.2 A Missing Constant

The breakdown of functional mapping occurs more profoundly in the relation between boundary and logical constants. This incompatibility is at the heart of the difference between boundary and Boolean systems.

Boundary	Boolean	Logical
	0	FALSE

Quite non-conventionally, boundary algebra assigns a semantic meaning to the absence of form. The particular mapping above is apparently not one-to-many, it is none-to-one. The gain in representational elegance, the underlying source of the one-to-many map, is that the boundary form for one of the Boolean constants does not exist. This however plays havoc with the conventional concept of functional mappings that underlie the morphisms of algebraic group theory.

3.1.3 No Functional Morphisms

Morphisms are structure preserving maps; a morphism between two systems is an identical structural shared by the systems. From a mathematical perspective, systems with morphic structures are indistinguishable with regard to that structure.

Since morphisms are functional maps, the one-to-many relation means that there can be no morphism from boundary to Boolean forms. The best one can do is to examine the many-to-one map from Boolean algebra to boundary algebra.

Morphisms can be represented as constraint equations and as diagrams. The rough idea is to show that the result of an operation in one system gives the same

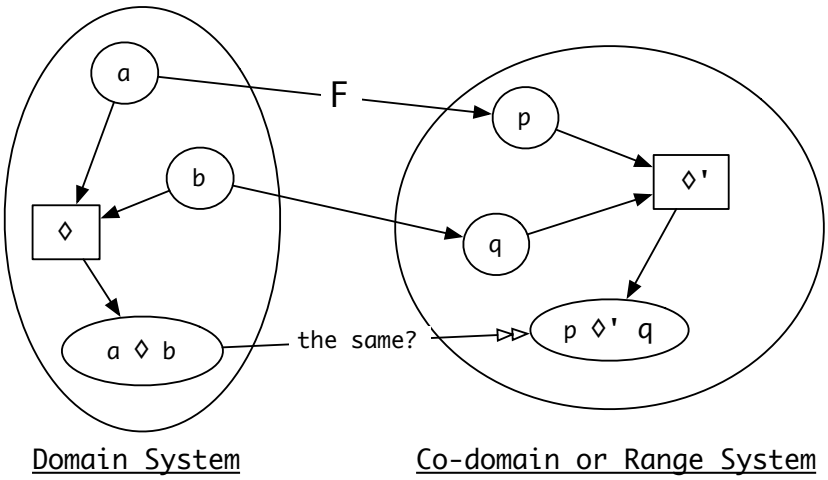
result as the corresponding operation in the other system. To do this, a morphism function is identified that maps between the two systems. Although the technique of morphisms is quite general, it is presented here using the specific structures of the two systems of interest, boundary and Boolean algebra.

3.1.3.1 The Morphism Constraint

The general constraint equation that defines a morphism is:

$$F[a \lozenge b] = F[a] \lozenge' F[b] \qquad \text{for every } a \text{ and } b$$

◊ is a generic binary function in the domain system while ◊' is the corresponding binary function in the system being mapped to (the co-domain or range). F is the morphism function, a function that maps one system onto the other. If F exists, then a morphism exists. Diagrammatically:



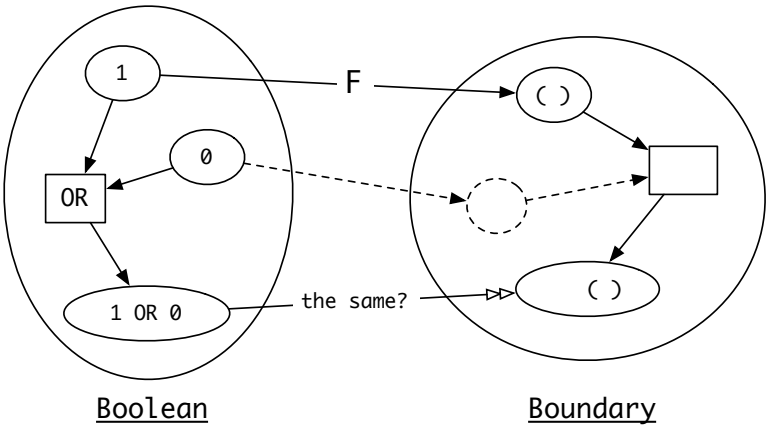
3.1.3.2 Boolean to Boundary

An essential distinction between boundary and Boolean algebra is the set of constants that each incorporates:

Boundary	Boolean	Logical
{ ( ) }	{ 0 , 1 }	{ FALSE, TRUE }

Boundary systems have one constant, the enclosure. Logic has two, the truth values, while two-element Boolean algebra has the binary numbers, {0, 1}. A morphism between two systems requires that each element in one system map to an element in the other system. Thus a morphism between Boolean and boundary systems cannot exist.

Consider two constants SHARING the same space. Pictorially, the dashed lines below indicate that there is nothing that the constant 0 maps to:

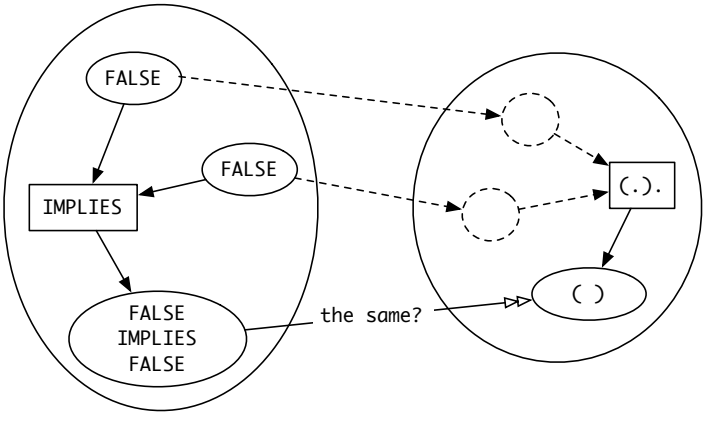


Since this example is limited to constants, it is not appropriate to use (( )), or any other void-equivalent, to stand in place of the non-concept "boundary false". Also note that the operator box for the boundary system is empty. There is no token that represents SHARING. SHARING is not an operator, it is an unstructured collection, a capability of the underlying substrate upon which forms are recorded.

Mark cannot share a space with Void, there is no Void element. Void is the substrate that supports marking, not an element in the boundary system. Void is not only outside the Mark, it is inside Mark as well. In the above diagram, the representation is explicit that the empty SHARING box is acting as a unary, not a binary operator. It is the Identity function. Were this diagram converted back into a morphism constraint, it would appear to redefine conventional logic:

$$F[1 \text{ OR } 0] \;=\; \text{Identity}[\, F[1] \,] \;=\; F[1]$$

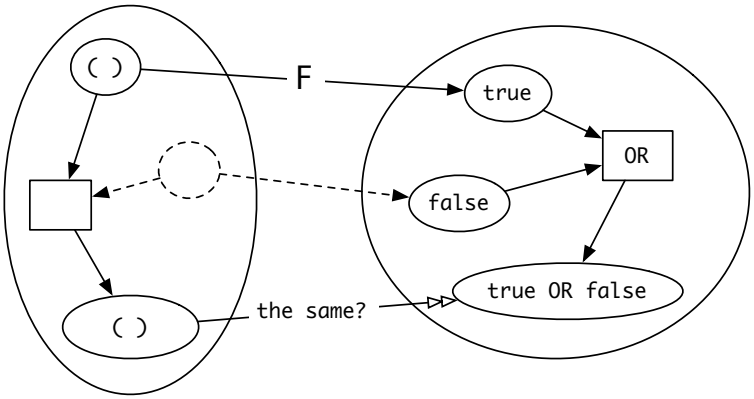
Consider the mapping of logical implication with two FALSE arguments:



In this case, not only does the constant FALSE fail to map, but the "operator" Mark converts into a constant.

3.1.3.3 Boundary to Boolean

A subtle distinction between a mapping and a function is that mappings are not required to cover the entire range being mapped to. It is permitted for the range to have extra, non-mapped elements. This suggests that the simple two-element case of mapping from boundary to Boolean systems be re-examined:



Here, the essential problem is that operations on the non-mapped constant are not permitted. Unfortunately, the two-element Boolean system cannot do without one of its two elements.

3.1.3.4 Patching the Mechanism

Who can blame a conventional mathematician who addresses these anomalies by providing a token for Void, say ⊥, and a sign for SHARING, say + ?

The problem is that this seemingly innocuous modification literally disassembles the boundary system, structurally converting it into a Boolean system. Putting something in place of nothing adds a new ground element to the boundary system. More significantly, it localizes Void to be in one specific place within the notation. Void pervades a boundary form, it is identified with the syntactic substrate (the paper, for example) and is thus everywhere. Labeling Void destroys its pervasive property, and consequently undermines the functionality of the rule of PERVASION.

Technically, since boundary algebra contains no functions at all, it is not an algebraic system. Rather, boundary algebra is a pattern-matching system based on a relational algebra of inclusion. Thus there is no functional morphism of any type between boundary and Boolean algebra. While Boolean algebra is a complemented, distributive lattice (and a lattice is a partially ordered set with upper and lower bounds), boundary algebra does not even manage to qualify as having a rudimentary algebraic structure (a set and a binary operation on that set). Boundary algebra is an algebra only in the broadest sense of a system that uses formal techniques to manipulate symbols.

The partial order relation that exists in both Boolean and boundary systems is discussed next.

3.1.4 Partial Order Relation

The mapping between boundary algebra and Boolean algebra is relational [Varela and Goguen(78)]. Both systems are partial orders. This is easily seen in boundary algebra, since BOUNDING (inclusion, enclosure, nesting boundaries) is the essence of ordering, while SHARING is the non-ordering structure that permits order to be partial.

A partial order (poset) is a relation that is reflexive, transitive and antisymmetric. For boundary algebra, the Mark is the binary order relation. What is inside is "less than or equal to" what is outside.

Conventionally, x ≤ y is read as "x is contained in y". The containment relation is explicit in a boundary representation:

x ≤ y =def= (x) y BOUNDARY

In comparison, the inclusion relation in Boolean logic is:

x ≤ y =def= x IMPLIES y BOOLEAN

3.1.4.1 Partial Order Morphism

A morphism of partial order is a mapping function between two posets. The morphism is defined as

x ≤ y = F[x] ≤' F[y] for all x, y

The morphism constraint from the Boolean to the boundary poset transcribes as:

x IMPLIES y = (F[x]) F[y]

The morphism function, F[x], simply maps Boolean to boundary constants, with the continuing absence of a mapping for 0:

Boolean	Logical	Boundary
0	FALSE	
1	TRUE	( )

The validity of this mapping can be checked by comparing the truth tables for IMPLIES in the Boolean logic system with the algebraic reduction results in the boundary system.

Below, a square bracket in boundary notation, [ ], is identical to a parenthesis, ( ); the square bracket is used solely for the meta-purpose of highlighting the boundaries that correspond to conventional logical connectives.

Boolean	Boundary	
FALSE IMPLIES FALSE = TRUE	[ ] = ( )	identity
FALSE IMPLIES TRUE = TRUE	[ ] ( ) = ( )	call
TRUE IMPLIES FALSE = FALSE	[( )] =	cross
TRUE IMPLIES TRUE = TRUE	[( )] ( ) = ( )	cross

3.1.4.2 Partial Order Properties

Boundary algebra can be employed to demonstrate directly that Mark has the partial ordering properties. Boundaries are reflexive, transitive, and antisymmetric. Conventionally, logical connectors are used to define relational properties, however the demonstration can be carried out purely within the boundary system. The partial order property definitions in each system follow, again the square bracket is used for highlighting logical transcriptions:

	Logical	Boundary
Reflexive	x ≥ x	(x) x
Transitive	x ≥ y AND y ≥ z IMPLIES x ≥ z	[(x) y] [(y) z] (x) z
Antisymmetric	x ≥ y AND y ≥ x IMPLIES x = y	[(x) y] [(y) x] x=y

The boundary algebra proofs that follow illustrate how boundary algebra can stand in place of logic. Three prerequisites are helpful:

1. A preliminary boundary theorem, Dominion, makes the proofs more direct:

( ) A = ( ) DOMINION

2. The boundary transcription of Boolean equivalence is needed:

x=y =def= (((x) y) ((y) x))

3. The boundary algebra treatment of the structure of proof using premises and conclusions is that BOUNDED premises share the same space as the conclusion. The literal boundary translation of the logical form of implication is:

Logic	Boundary
A IMPLIES C	[A] C

In the case of a conjunction of premises, the literal transcription immediately reduces via INVOLUTION:

A AND B IMPLIES C	[[[A] [B]]] C	
	[A] [B] C	inv

Each premise is individually BOUNDED to express the structural relation that premises are less than the conclusion. That is, a valid conclusion pervades its premises.

Now the boundary algebra proofs that a boundary is a partial order. Definitions that reduce to Mark are logically TRUE.

Reflexive	(x) x		def
	( ) x		per
	( )		dom
Transitive	[(x) y] [(y) z]	(x) z	def
	[y] [(y)]	(x) z	per
	[y] [ ]	(x) z	per
	[ ]		dom
Antisymmetric	[(x) y] [(y) x]	x=y	def
	[(x) y] [(y) x]	(((x) y) ((y) x))	equiv
	[(x) y] [(y) x]	( )	per
		( )	dom

3.2 Other Candidates for the Essential Difference

There are several potential candidates for the primary differentiator between boundary and Boolean algebra. Brief discussions of other differentiators of interest follow. These include specific conventional characteristics that boundary forms lack, specific characteristics of boundary forms that conventional expressions lack, specific operational characteristics that are not available in conventional computation, and conceptual tools that boundary systems contribute to mathematical thought.

3.2.1 Absence of Group Theoretic Structures

Boundary algebra does not include the conventional structural concepts of cardinality, associativity, commutativity, and arity. Underlying the absence of structural properties associated with functions is a fundamental observation: there are no functions in boundary algebra.

Conventional properties could be added to boundary algebra as elaborations, by introducing constraint equations on boundary forms. It is common practice to construct systems incrementally; for example, adding the rule of commutativity to a simple group makes it an Abelian (commutative) group. Boundary algebra can, then, be viewed as a minimal system upon which to add structural constraints such as cardinality and commutativity.

It is inappropriate to say that SHARING is an "associative, commutative function". These properties are not possible in a void-substrate space, since by definition, Void supports no properties. It is inappropriate to say that BOUNDING is a "binary function" since, due to PERVASION, Mark distinguishes the SHARING space inside, without influencing the pre-existent SHARING space outside. It is not even possible to write the rule of PERVASION as a conventional function.

Seen from this perspective, group theoretic properties, such as commutativity and right- and left-identities, have a strangely syntactic origin arising from writing expressions on a line.

3.2.2 Semantic Void/Single Constant

Due to the semantic void, boundary algebra is conceptually leaner that conventional Boolean algebra. It is an error to force Boolean concepts onto boundary systems, such as the idea of marking Void with an indicator, say 0 or the empty set, { }. To understand boundary math, it is necessary to adopt different ideas, not to eliminate new ideas through unnecessary elaboration.

The notion of a semantic substrate is crucial to understanding boundary mathematics. Drawing a Mark on top of the void substrate creates one distinguished space, the inside of the Mark, the other (pre-existing) distinguished space is the entire page upon which the Mark rests.

One source of confusion is the interpretation of Mark as a generator of duality, making inside and outside mutually exclusive. Void can be understood in a more natural way as a pervasive space. The inside is separate from the outside, but the outside is not separate from the inside. In this way, Marks are semipermeable; from the outside, a Mark does not present a barrier.

Void works like physical space; a solid object distinguishes the space it occupies, but it does not "push aside" or exclude that space. Similarly, a Mark distinguishes the page upon which it is recorded, but it does not cut the page into physically separate pieces.

The value, or meaning, of Void is inverted by placing a Mark upon it.

---> ( )

There is no token that can mean Void, since any representation of Void denies its meaning, in a very literal sense. The value of a representation of Void is Mark, not Void. The explicit representation of Void is a semantic error. Since Void is unique, multiple representations of Void are multiple errors.

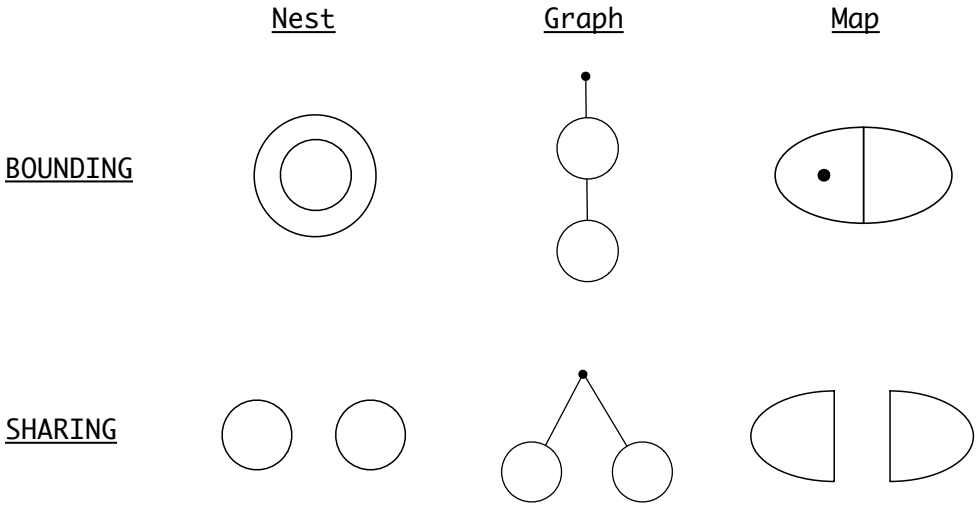
The idea that one Boolean constant can be omitted from Boolean notation without loss of expressive power is an essential innovation. Two-valued Boolean algebra is over-specified, the second value can be inferred from the absence of the first. Although the syntactic structure "FALSE" is not present, the semantic idea of FALSE is explicitly distinguished as the empty content of an empty Mark, and is implicitly present prior to any Mark as the substrate for marking.

3.2.3 Inside and Outside of Forms

Conventional linear notation treats tokens (for example: foo, +, $) as solid atoms, providing an outside but not an inside. Set theoretic curly braces, {...}, provide an inside but not an outside. Boundary mathematics is unique in that constants and compound forms have both an inside and an outside. This requires a spatial rather than a linear notation.

Delimiting parentheses (that is, matched but potentially separated token pairs) play a special role in string languages. The difference between context-sensitive and context-free languages is the presence of paired tokens that permit an inside and an outside. In addition, parentheses are the minimal string structure that requires a memory in pushdown automata.

The idea of containment can be represented in ways other than nested boundaries. It can also be expressed as a graph, for which links represent containment relations [Bricken(86)], or as a map, for which common borders represent containment relations. Maps require a locator to identify the outermost region that is overt in nests, graphs and text. Thus, in different pictorial notations, enclosure (nesting) is also connection (graph) is also contact (map):



3.2.4 Spatial (Non-Linear) Notation

The characteristics of a spatial notation are fundamentally different that those of a textual, linear notation. Above, three varieties of planar representations are illustrated. Each is a topological variant; new spatial notations are generated by geometric and topological transformations, not by substitution and rearrangement of tokens.

Parenthesis strings have been studied extensively as Dyke languages and as interpretations of Catalan numbers. There are dozens of known isomorphic mappings between balanced parenthesis strings, various number sequences, and spatial forms. The spatial representations include binary trees, trivalent planar trees with a single root, lattice paths, silhouettes of mountain ranges, non-intersecting chords joining points on the circumference of a circle, specific types of poset diagrams, and stacks of coins with a contiguous bottom row.

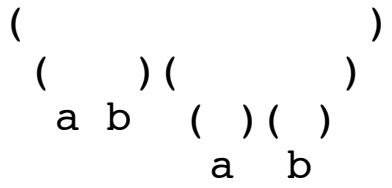
Spatial notation permits representations that are typographical (parentheses), planar (paths, nestings, networks, maps) and manipulable in physical space (steps, cubes, rooms). All support formal computation through pattern-matching and substitution using the three algebraic boundary rules. Conventional mathematical notation has nothing similar.

3.2.4.1 Topological Variants

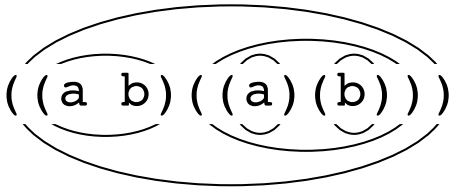
The following vertical transformation sequences illustrate two families of spatial notation generated by structural variation of the parenthesis syntax.

((a b)((a)(b)))

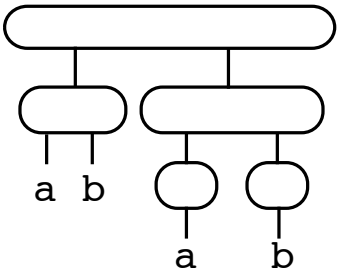
Textual parenthesis string



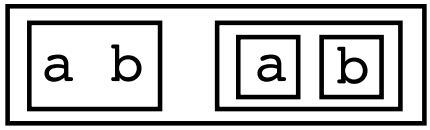
Extruded



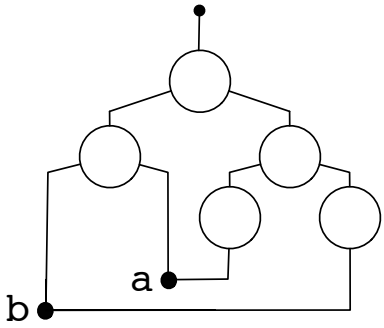
Capped



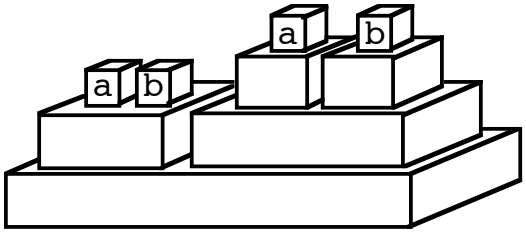
Tree



Boxes



Network



Stacked Blocks

Stacked Blocks are generated by rotating the Box representation 90° out of the plane of the paper and extruding. This 3D representation is of particular interest: when interpreted as logic, it provides a tactile, manipulable form for deduction. Logical reasoning can be concrete as well as abstract.

3.2.4.2 Notational Variety

To illustrate the wide diversity of spatial notations, several other representations of the above form, ((a b)((a)(b))), follow:

1. Distinction Network: this graph reduces/evaluates via local asynchronous message passing [Bricken(95)].

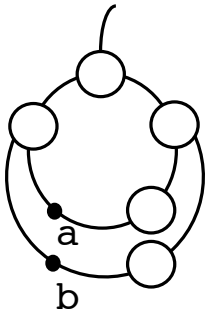
2. Distinction Steps: steps are generated by rotating the distinction network out of the plane of the paper by 90°, modifying the size of each node so that they overlap, and looking down.

3. Crossbar Switch: a variety of electronic circuit based solely on wires and inverters; bars are Marks.

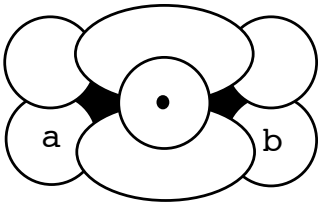
4. Distinction Path: all Marks have been unified into the single boundary that the computational path crosses and re-crosses.

5. Distinction Rooms: doors represent semipermeable boundaries. Computation can be achieved by walking through rooms and closing doors.

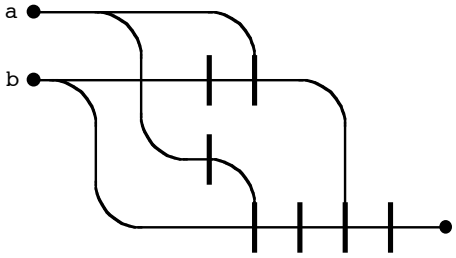
6. Circuit Schematic: conventional electric circuit diagrams use a spatial notation that evaluates via signal propagation.



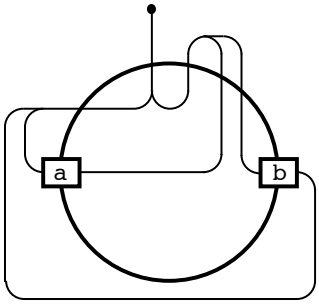
Distinction Network



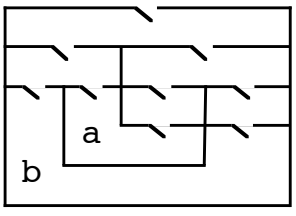
Distinction Steps



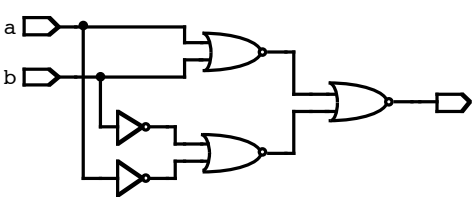
Crossbar Switch



Distinction Path



Distinction Rooms



Circuit Schematic

3.2.5 Void-Equivalence

Any arbitrarily complex configuration of boundaries will reduce to either Mark or Void when Spencer-Brown's arithmetic rules are applied to delete structure. For example:

```
( (( )) (( )( )( )) )                boundary form
(       (( )( )( )) )                cross
         ( )( )( )                    cross
         ( )                          call
```

The essential structure of a form is a single Mark, together with meaningless void-equivalent "junk" that exists syntactically but not semantically. Boundary arithmetic computation is simply a process of deleting the junk, with the possibility that nothing, not even a Mark, will remain. Unlike all explicit conventional expressions, void-equivalent boundary forms do not have a value. Technically void-equivalent forms lose all characteristics upon evaluation, since their only characteristic is their meaningless syntax.

Each boundary algebra rule permits construction of void-equivalent forms by application from right-to-left. Since these rules can be applied any number of times, Void is teeming with virtual structure. During proofs, constructive rule applications are marked by the + sign. An example constructive sequence:

```
                                      absence of form
(( )  )                               cross+
(( ) z)                               occ+
((z) z)                               per+
```

Void-equivalent forms can quickly become complex. Consider the boundary DISTRIBUTION theorem (proved in Section 4.2):

```
A ((B)(C))  =  ((A B)(A C))              DISTRIBUTION
```

Substituting each side of the equation into the void-equivalent construction yields:

```
((    z     )       z     )
((A ((B)(C))) ((A B)(A C)))               subst
```

The resulting form is no longer obviously void-equivalent. This construction from nothing illustrates on why Boolean algorithms are intractable, and how PERVASION can be used to disentangle void-equivalent forms.

Void-equivalence sheds light on an essential characteristic of logical computation. An explicit syntactic form consists of one kernel of information

(existent as Mark, or non-existent as Void) that is confounded with other void-equivalent structure. Logical computation is removing the irrelevancies. Conventional logic incorporates ideas of false information, material implication, multiple syllogistic structures, inference rules, replacement rules, and proof strategies; through the lens of boundary algebra, each can be seen as an unnecessary complexity.

3.2.6 Object/Operator Unification

Boundary forms are structural objects. BOUNDING acts to distinguish what is inside, a boundary is a relation between inside and outside. Conventional mathematics requires unique tokens that distinguish objects, operators, and relations. However, in a computation, what is operator and what is object is determined by the implementation mechanism.

Boundary reduction rules can be implemented by parallel pattern-matching or as an asynchronous network process. Rules can also be implemented by a variety of other means, such as string rewriting, tree traversal, and finite state machines. However, it is useful to keep in mind that implementation languages characterize the computational machine rather than the mathematics.

In a conventional procedural computer language, operations are instructions that operate on data. In a functional regime, everything is an operation, data is simply what is left unevaluated. In a pattern-matching regime, operations are subsumed by the pattern-matching engine; all forms are objects. In an asynchronous network reduction regime, objects and operators are subsumed by the network structure; operations become messages between nodes in the network.

Object/operator unification in boundary systems is deeper than an implementation mechanism. Boundary algebra variables stand in place of zero, one or many forms; they abstract any collection of forms sharing a space. The space shared by forms does not support grouping or ordering. Thus, without concepts of necessary existence, cardinality, arity, associativity, and commutativity, the conventional distinction between operator and object lacks support.

Consider the interpretation of BOUNDING as a relation between inside and outside. How would the form ((a)(b)) be written in a relational syntax? The transcription to the relation R[inside, outside] might be:

((a)(b)) =transcribe= R[R[a, R[b]]]

Here, the outer boundary has no explicit outside object, the boundary is a unary operator. One of the two inside forms, (a) and (b), must be read as unary in order for the other to be binary. When commutativity is added, both boundaries can chosen to be either a binary relation or a unary function.

The net result is that what is being represented by nested containment is not the same as the relation R[inside, outside], The boundary relation undermines the SHARING operation, while SHARING undermines the relational interpretation. It is better to accept the non-conventional boundary pattern interpretation that to attempt to fit boundary forms into conventional functional structures.

3.2.7 Semipermeable Boundaries/Operational Transparency

A boundary cannot be strictly interpreted as a function or as a relation. Difficulty arises when considering how PERVASION works over nested boundaries. Assume that the boundary is contained by an outer SHARING space (its context) and contains an inner SHARING space (its content). Consider the functional interpretation of BOUNDED-SHARING, (A):

s[A, B] =transcribe= (A B)

The rule of PERVASION gives permission to alter the arguments of a remote nested function, while ignoring intervening functions that contribute to the remote argument. Importantly, PERVASION does not mean that steps are taken across each intervening boundary until a pattern-match is found, it means that intervening boundaries are transparent. From the outside, there are no intervening boundaries. Consider this example:

```
s[ s[a,b], s[a, s[b, c        ]]] =?= ( (a b) (a (b c      )) )    eg
s[ s[a,b], s[a, s[b, c, s[a,b]]]] =?= ( (a b) (a (b c (a b))) )    per+
s[ s[a,b], s[a, s[b, c, s[   ]]]] =?= ( (a b) (a (b c (   ))) )    per
s[ s[a,b], s[a                ]] =?= ( (a b) (a            ) )    occ
```

The second line uses PERVASION to insert a form deep into another form. Under the functional interpretation, this is equivalent to adding a new argument to a remote function. The third line uses PERVASION from two different functions to extract both arguments of the innermost function, converting it into a constant. The fourth line uses OCCLUSION to remove a compound argument, changing the host function from binary to unary.

Naturally, permeable function boundaries, modification of remote function arguments, dynamic arity, and transparent functional dependencies are foreign to conventional mathematical systems.

3.2.8 Catalytic Computation

Void-equivalence can be used in computation constructively as well. Forms outside a boundary are everywhere inside the boundary as virtual forms. Virtual computation gives permission to postulate, or imagine, void-equivalent forms

pervading the substrate upon which forms are recorded (e.g. the page). Virtual forms are not recorded, they are not syntactically explicit. Rather they are used as hypotheses that can be discarded at will.

Interaction of void-equivalent forms with explicit forms leads to the catalytic identification of void-equivalencies within the syntactically explicit, and thus to reduction via deletion. Virtual forms that are useful for reduction can be postulated without concern; virtual forms that may not be useful can be forgotten (cast into the void) at any time.

The reduction example above, ((a b) (a (b c))), is repeated here using virtual forms. A virtual form is indicated by carets, ^...^. The reduction shows virtual structure explicitly, however the carets are meant to indicate that the presence of the virtual form is considered, not necessarily recorded.

```
( (a b       ) (         a (         b c)) )        boundary form
( (a b       ) ( ^(a b)^ a ( ^(a b)^ b c)) )        virtual per
( (a b       ) ( ^(  b)^ a ( ^(   )^ b c)) )        per
( (a b       ) ( ^(  b)^ a               ) )        occ
( (a b       ) (         a               ) )        forget virtual
( (a b ^(a)^ ) (         a               ) )        virtual per
( (a b ^( )^ ) (         a               ) )        per
(              (         a               ) )        occ
                         a                          inv
```

Therefore, by following rules to convert one form into another,

((a b) (a (b c))) = a

Boundary algebra as well provides a body of unique, non-conventional void-based transformation and solution techniques.

3.2.9 Thinking with Boundary Concepts

Boundary rationality suggests that it is possible to reason rationally without using the logical connectives. It is an extremely difficult and bold step to realize that conventional rationality can be carried forth in thought and in computation without using the familiar concepts of AND, OR, IF, and NOT.

People have tremendous difficulty using logic correctly. For all but the specifically trained, logic is puzzling. Thinking in logic creates an unnecessary cognitive load. In contrast, deletion of void-equivalent forms, although unfamiliar, is at least conceptually simple. Learning to think with boundaries rather than with conventional logic requires considerable effort; this paper merely establishes that such thinking is both possible and different.

Imagine a computer programmer writing a COND statement (nested IF-THEN-ELSE):

```
IF A THEN B ELSE
     IF C THEN D ELSE
          IF E THEN F ELSE
               ...
                    IF TRUE THEN H
```

The use of COND requires structured planning, prioritizing, testing, and terminating, all revolving around the semantic use of IF-THEN-ELSE. The boundary form of COND looks like this:

```
(  ((A) B)
  (  ((C) D)
    (  ((E) F)
      (...
        ((( )) H) ...))))
```

The above formatting carries residual structure from the direct logical transcription. A different alignment shows the same structure in a different way. Below, the void-equivalent form (( )) is also eliminated.

```
(      (B (A))
 (     (D (C))
  (    (F (E))
   (...
        (H    ) ...))))
```

In formulating and using a boundary COND, the thought process does not include IF-THEN-ELSE. That is, the pattern of the parenthesis structure for COND achieves the same result but using words like "outside", "bounded" and "pervades". Boundary COND might be verbalized as something like this:

```
Outermost, B pervades A is bounded.
  This form pervades other structurally identical forms
    Repeated at each depth.
```

Thinking in boundary logic might also begin in the deepest space, the space that contains H. All other forms pervade this space, so that everything outside is available to it. The deeper the nesting, the less likely that the nested forms will influence the outcome.

In sum, there is an efficient way to think using boundary concepts natively. The conventional words/concepts one uses to construct things like a functional COND statement embed redundancies not present when thinking in a spatial nesting language.

4 BOUNDARY ALGEBRA IS NOT LIMITED TO TWO-VALUED BOOLEAN ALGEBRAS

Boundary algebra can be interpreted as the algebra of subsets. The algebra of finite subsets is a Boolean algebra based on the powerset of a set of atomic elements, the binary operations of union and intersection, and the set complement unary operation. The zero element is the empty set, { }, and the unit element is the Universe, X, the set of all elements. Since the algebra of subsets is isomorphic to all other Boolean algebras, this demonstration suffices to show that boundary algebra applies to Boolean algebras in general.

4.1 Mapping the Algebra of Subsets onto Boundary Algebra

As would be expected, the mapping from the algebra of subsets to boundary algebra is non-standard. In particular, the zero element, { }, is Void, rendering it non-existent; the unit element, X, is Mark; union is SHARING; and intersection is a compound operator consisting of one SHARING and three BOUNDINGS. The functional mechanisms of the algebra of subsets are expressed solely by boundary relations.

	Algebra of Subsets	Boundary Algebra
Zero	{ }	
Unit	Universe, X	( )
Complement[A]	A' = X - A	(A)
Union[A, B]	A ∪ B	A B
Intersection[A, B]	A ∩ B	((A)(B))

4.2 Proof of the Rules of Boolean Algebra

Each of the constraint equations for Boolean algebra has an analog in boundary algebra. In the table below, the generic operator token, ◊, stands in place of both binary operators, union and intersection.

	Algebra of Subsets	Boundary Algebra
Associative	(A ◊ B) ◊ C = A ◊ (B ◊ C)	no grouping concept
Commutative	A ◊ B = B ◊ A	no ordering concept
Zero/unit	A ∪ 0 = A A ∩ 1 = A	INVOLUTION ((A)) = A
Complement	A ∩ A' = 0 A ∪ A' = 1	OCCLUSION (( ) A) =
Distributive	A ◊ (B ◊' C) = (A ◊ B) ◊' (A ◊ C)	PERVASION A {A B} = A {B}

To show coverage of the algebra of subsets by boundary algebra, it is sufficient to prove each of the Boolean constraint rules using the three rules of boundary algebra.

Associativity — Subsumed by space without structure.

Commutativity — Subsumed by space without metric.

Zero/unit

A ∪ 0 = A

A = A

A ∩ 1 = A

((A)(( ))) = A — transcribe
((A)) = A — inv
A = A — inv

Complement

A ∪ A' = 1

A (A) = ( ) — transcribe
A ( ) = ( ) — per
( ) = ( ) — dom/occ

A ∩ A' = 0

((A)((A))) = — transcribe
((A)( )) = — per
= — dom/occ

Both distributive rules in the algebra of subsets can be proved using boundary algebra DISTRIBUTION:

A ((B)(C)) = ((A B)(A C)) — DISTRIBUTION

Boundary DISTRIBUTION is a theorem that can be derived using only the three initial equations of boundary algebra. The following boundary proof nicely illustrates the fundamental role of PERVASION.

```
   A                        ((  B)(  C))          lhs
   A                        ((A B)(A C))          per+
( (A)                     ) ((A B)(A C))          inv+
( (A) ((    A B)(    A C)) ) ((A B)(A C))          per+
( (A) (((A) A B)((A) A C)) ) ((A B)(A C))          per+
( (A) ((( ) A B)(( ) A C)) ) ((A B)(A C))          per
( (A) (                  ) ) ((A B)(A C))          occ
                             ((A B)(A C))          occ, rhs
```

Distribution, union over intersection:

$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$$

```
A   ((B)(C))  =  ((A B)(A C))              transcribe, dist
```

Distribution, intersection over union:

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$$

```
  ((A)(B C))  =  ((A)(B)) ((A)(C))         transcribe

  ((A)(      B        C   ))               lhs
  ((A)( (   (B)) (   (C)) ))               inv+
  (   ( ((A)(B)) ((A)(C)) ))               dist
        ((A)(B)) ((A)(C))                  inv, rhs
```

Thus the Boolean algebra rule of Distribution is not fundamental, it is a theorem of PERVASION.

4.3 One- and Two-element Algebra of Subsets

The powerset, PS, for a two-valued Boolean algebra has four members:

$$PS[\{0, 1\}] = \{ \{ \}, \{0\}, \{1\}, \{0,1\} \}$$

These four members can be read as "nothing" (the empty set), the two elements, and "everything" (the Universe).

In contrast, the powerset for the one-valued boundary algebra has two members:

$$PS[\{ ( ) \}] = \{ \{ \}, \{ ( ) \} \}$$

which again can be read as "nothing" (Void) and "everything" (Mark). This mapping suggests that boundary arithmetic can be interpreted as a Calculus of Universes [Bricken(93)].

There is clearly no Boolean algebra of one element. The rules of Boolean algebra call for a zero and a unit element, that is, for at least two elements. As well, the operators union and intersection are binary. The only way a one element algebra can satisfy a binary operator is through the degenerate case of the union or intersection of an element with itself.

The foundational role of boundary mathematics in elementary set theory is evident: boundary arithmetic is a one-element algebra of subsets that covers the two-element algebra of subsets.

4.4 The Four-element Algebra of Subsets as an Example

The algebra of subsets with four elements illustrates the rules of boundary algebra as they apply to operations in an n-element algebra of subsets.

The powerset for a four-valued Boolean algebra consists of sixteen elements:

PS[{a,b,c,d}] = { { }, {a}, {b}, {c}, {d},
{a,b}, {a,c}, {a,d} {b,c}, {b,d}, {c,d},
{a,b,c}, {a,b,d}, {a,c,d}, {b,c,d}, {a,b,c,d} }

In using boundary algebra to compute results for the algebra of subsets, a set, S, is represented as the elements of the set sharing space. The curly braces and the commas separating members in set notation are both deleted:

S = {a, b, c, d} =transcribe= a b c d

4.4.1 Definitions vs Algorithms

A preliminary distinction between definitions and algorithms is necessary. The algebra of subsets defines operations such as compliment, but it does not include algorithms for computing the results of operations. For example, it is easy to see that b is the only common element in this set intersection:

{a, b, c} ∩ {b, d} = {b}

However, there is no specification as to how to arrive at this conclusion computationally. One algorithmic approach would be to take each element of the smaller set {b, d} and in turn compare it to each element of the larger set, saving those elements that are found in both sets.

In contrast, the three void-equivalent rules of boundary algebra do specify steps to take in computing a result. An algorithm is still needed to apply the rules, but it can be the same pattern-matching algorithm that applies those rules to any interpretation of boundary algebra.

Consider the union operation in each system:

```
{a, b} U {b, c}  =  {a, b, c}
 a  b      b  c                          transcribe
 a  b         c                          per
```

The set theoretic result is simply the result of a definition, while the boundary algebra result is computed by one application of PERVASION.

Unlike the algebra of subsets, the boundary algebra union operation is not binary, it applies to an arbitrary number of sets. As an example:

```
{c} U {a, b} U {b, c} U {a, c} = {a, b, c}
 c      a  b     b  c      a  c             transcribe
 c      a  b                               per
```

4.4.2 Interpreting the Boundary as Set Complement

For the algebra of subsets, the boundary is interpreted as a set partition, the compliment operation.

Boundary	Subset	Four-element Example
	{ }	{ }
( )	X	{a,b,c,d}
(A)	A'	{a,b,c}' = {d}

The empty set, { }, is unique and is a member of every set. It therefore shares some of the characteristics of Void. There is little to change in casting the empty set into the void; in conventional practice, the empty set is used primarily as a marker for nothing.

The Universe, however, has a particular interpretation when it is instantiated as a particular set of elements. Mark inherits that specific interpretation. In the four-element example, Mark is the four elements sharing space, while the complement of Mark is the empty set, the four elements bounded:

( ) = a b c d = X

( )' = (a b c d) = { }

Which version of Mark to use depends on the purpose of a computation. For manipulation of abstract rules, Mark can be used, but for specific computations over specific sets, the explicit collection of elements must be used. At any time, each can be substituted for the other with no loss of correctness.

For an example, consider the set difference operation, A - B, which is defined as those elements in A but not in B.

$$A - B = A \cap B'$$

Transcribing into boundary algebra:

```
A - B  =  ((A)((B)))
          ((A)  B  )                        inv
```

The complement is the set difference between a set and the Universe:

```
B'  =  X - B  =  ((( )) B)
                 (      B)                 inv
```

Using the abstract Mark as Universe generates a definition of the complement of B as (B). However, consider computing a specific complement:

$\{a, d\}' = X - \{a, d\} = \{a, b, c, d\} - \{a, d\} = \{b, c\}$

```
(a  d)  = ( ) - a  d  =  a  b  c  d  -  a  d          hybrid
```

Reducing the boundary definition of set difference,

```
(a  d)  =  ((a  b  c  d) a  d)            def ((X) B)
           ((   b  c   ) a  d)             per
```

arrives at an intermediate structure:

```
(a d)  =  ((b c) a d)
```

Here, the computation of the complement is not yet completed, but note that the computed complement is part of the right-hand-side. The specific complement is the solution to the above equation. This solution, (a d) = b c, can be verified by substitution:

```
(a d)  =  ((b c)   a d  )
(a d)  =  ((b c) ((a d)))                  inv+
 b c   =  ((b c) ( b c ))                  subst
 b c   =  ((b c)       )                   per
 b c   =    b c                            inv, identity
```

It is never necessary to actually solve the equation, since the solution is embodied directly within the right-hand-side. This can be demonstrated using void-equivalence:

```
(a d)  =  ((b c) a d)
(a d)  =  (      a d)                          identity
           (b c) =
            b c  =  ( )
```

This solution makes a subtle redefinition of the Universe by removing {a, d} from all forms. The particular solution to the intersection equation is "everything" permitted by the structural constraint defined by the equation.

In practical terms, the form of the complement is one level of nesting deep, while the computed complement is two levels of nesting deep.

4.4.3 The Boundary Form of Intersection

Finally, consider the boundary form of the set intersection operation. Again, boundary algebra intersection is computed using the three initial rules, and again boundary intersection is not binary, the computation applies concurrently to any number of subsets.

Consider the following computational example:

```
 {a, b, c} ∩ {b, c} ∩ {b, d}  =  {b}

((a  b  c)    (b  c)   (b  d))              transcribe
((a     c)    (   c)   (   d))     b        dist
```

Boundary DISTRIBUTION, like other boundary operators, applies to any number of component forms, it has no specific arity.

The intermediate form above includes the computed result, b, within it. One way to understand this is to appeal to the definition of the form of intersection,

```
 ((X)(Y)) =              when X, Y have no common members
```

Therefore,

```
((a     c)    (   c)    (   d))  b          dist
                                 b          void-equivalent
```

There is a short cut. Once boundary DISTRIBUTION has been applied to a form, the remaining nested form contains no replicates. Within the interpretation for

subsets, a form in the pattern of intersection with no replicates is always void-equivalent.

Finally, consider an example of the decomposition of the form of intersection into unions and complements:

```
 {a, b, c} ∩ {b, d}

((a  b  c)    (b  d))                    transcribe
(    d        (b  d))                    {a,b,c}' = {d}
(    d         a  c )                    {b,d}' = {a,c}
          b                              {d,a,c}' = {b}
```

At any step of this computation, a boundary algebra rule could be applied. For example, at the second and fourth steps:

```
(    d        (b  d))                    {a,b,c}' = {d}
(    d        (b   ))                    per
(    d        a c d )                    {b}' = {a,c,d}
(    d        a c   )                    per
          b                              {d,a,c}' = {b}
```

Further, the specific definition of complement for any bounded form can be substituted at any time, thereby computing each complement in the decomposition.

```
((          a b c)  (          b d))     transcribe
(((   X   ) a b c)  ((   X   ) b d))     subst def
(((a b c d) a b c)  ((a b c d) b d))     subst Universe
(((      d) a b c)  ((a   c  ) b d))     per
(        d            a   c        )     form of complement
        ((   X   ) d a c)                subst def
        ((a b c d) d a c)                subst Universe
        ((  b    ) d a c)                per
            b                            form of complement
```

The boundary result is then interpreted as the computed intersection in a four-element algebra of subsets:

```
((a  b  c)    (b  d)) = b
 {a, b, c} ∩ {b, d}  = {b}
```

Boundary algebra forms are never more than three levels deep when interpreted for the algebra of subsets, making PERVASION a shallow transformation. In contrast, boundary forms interpreted for logic (a two-element algebra of subsets) can be deeply nested. This suggests an as yet unexplored conventional technique of assigning a semantics to deeply nested sets.

5 BOUNDARY ALGEBRA DOES NOT HAVE GROUP STRUCTURE

The evolution of mathematics has embraced Boolean algebra very deeply; nearly all mathematical approaches assume, often implicitly, the conceptual structure embodied in logic. Boundary algebra, then, should be expected to provide new concepts and ideas for many mathematical systems. Consider group theory as an example.

Group theoretic systems include a set of objects, S, in the particular system, and at least one binary operator, ◊ . The set must be closed under the operator; that is, applying the operator to members of the set yields another member of the set. Group theory identifies a hierarchy of types of algebraic systems. Each type of group theoretic system has its own type of morphism, or sameness.

Many published articles on boundary algebra claim to demonstrate that boundary and Boolean algebras are isomorphic. Isomorphism is the strongest type of structural identicality, asserting that the difference between the two systems is solely one of notation. The sequel shows that boundary and Boolean algebra are structurally different to an extreme degree, that the only way the two systems can be seen to be morphic is to redefine one of them to actually be the other system. Claims of boundary/Boolean isomorphism are based on the logical fallacy of circular reasoning.

5.1 Types of Structure

Group theory classifies types of mathematical systems using a hierarchy of accumulated properties:

System name	Property	Constraint equations
groupoid	< S, ◊ >	◊ is binary and closed
semigroup	associative	$(a \lozenge b) \lozenge c = a \lozenge (b \lozenge c)$
monoid	identity	$i \lozenge a = a \lozenge i = a$
group	inverse	$a \lozenge a^{-1} = a^{-1} \lozenge a = i$
Abelian group	commutative	$a \lozenge b = b \lozenge a$

5.1.1 Groupoid

Groupoids have minimal structure: a set and a closed binary operation.

The boundary algebra candidates for groupoid operations are:

Possible Operation	Example	Functional Notation
BOUNDING	(a b) c d	B[in, out]
BOUNDED-SHARING	(a b c)	B[in, Void]
SHARING	a b c	S[space]

Each of these operations is closed, however all incorporate the problematic Void. An operation that uses or results in Void is technically not closed.

None of these operations is strictly binary. A space, whether bounded or not, can support any number of elements; there is no mechanism, for example, to group or order three forms sharing the same space. As well, all varieties of BOUNDING address a SHARING space:

```
BOUNDING         =interpret=  B[S[in], S[out]]
BOUNDED-SHARING  =interpret=  B[S[in], S[ ]]
```

In all cases, SHARING is a necessary boundary algebra candidate for the group theoretic operator. BOUNDING interpreted as a binary operator on two SHARING spaces will also be considered. The difficulty with BOUNDING is that deeply nested contents support PERVASION, making nested boundaries transparent and undermining the idea of specific function arguments.

These difficulties can be finessed by relaxing the constraint that the operator is a binary function (or a function at all), and by accepting the dubious closure of Void. More radically, it is necessary to ignore the fact that, in boundary arithmetic, the set S has only one member.

5.1.2 Semigroup

A semigroup is a set S and a binary operation ◊ that is closed and associative. Associativity is conventionally written as:

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

BOUNDING is not associative:

```
B[B[1,2],3] = ((1) 2) 3
B[1,B[2,3]] = (1) (2) 3
```

The associativity of SHARING cannot be represented within a boundary system:

```
 a   b    c  =  a    b   c
```

Using INVOLUTION, it is possible to collect boundary elements in different spaces without modifying the value of an expression:

```
((a   b))  c  =  a  ((b   c))
```

What is lost in this approach is the meaning of SHARING, the operator is no longer simple, it is a compound construction of one SHARING and two BOUNDINGS.

The associativity property can be represented in diagrams with arrows. Associative means that mapping arrows between pairs of elements can be followed in any order. In the diagrammatic format, the same result, g, can be reached by different paths. For {a, b, c, d, e, g} in S:

```
a ◊ b ---> d                      e <--- b ◊ c
           d ◊ c --->  g  <--- a ◊ e
```

SHARING could be considered as a (pseudo)binary boundary operator that is associative. Using the diagrammatic format:

```
a   b ---> d                      e <--- b   c
           d   c --->  g  <--- a   e
```

The semi-group abstract operator ◊ can be loosely interpreted in both boundary and Boolean formalisms. The Boolean interpretation is natural, the boundary interpretation requires a redefinition of SHARING. The redefinition is accepted in order to continue.

5.1.3 Monoid

A monoid is a semigroup that has an identity element, i, where identity is defined as

```
a ◊ i  =  i ◊ a  =  a
```

In the boundary system, Void is the identity for SHARING:

```
a   i  =  i   a  =  a                    transcribe
a      =      a  =  a                    void-substitution
```

SHARING is a monoid when the identity element is Void. This remains problematic, since the boundary identity (non)element, Void, does not exist.

5.1.4 Group

A group is a monoid for which every element, a, has an inverse, a^{-1}. That is, S can be separated into two subgroups by pairing elements with their inverses.

The inverse of an arbitrary boundary form is that form BOUNDED:

Element	Inverse
	( )
( )	(( ))
A	(A)

Every element clearly has an inverse by construction, however, one inverse does not have an element.

The constraint equation for inverses is:

$$a \lozenge a^{-1} = a^{-1} \lozenge a = i$$

Transcribing into boundary form, a^{-1} is (a), ◊ remains SHARING.

```
a  (a)  =  (a)   a   =                    transcribe
a  ( )  =  ( )   a   =                    per
   ( )  =  ( )     =!=                    dom
```

When the operator is SHARING, the identity element fails to match the composition of element and inverse-element. A moderately redefined SHARING forms a monoid, but no degree of redefinition can make SHARING into a group.

Considering Mark rather than Void as the identity element supporting inverses only shifts the contradiction to the definition of identity:

$$a \lozenge i = i \lozenge a = a$$

```
a   i  =  i   a  =  a                     ◊ = SHARING
a  ( ) = ( )  a  =  a                     subst i = ( )
   ( ) = ( )    =!= a                     dom
```

5.2 Compound Configurations

BOUNDING and SHARING both fail as group operators; is there a compound boundary configuration that can serve as a group function with an identity and an inverse? Since compound operators inherit properties from simple operators through composition, the answer is no.

For illustration, the potential group structure of a compound form is presented next. Consider the compound pseudo-binary boundary pseudo-operator: ((.)(.))

The closure of ((.)(.)) is obvious, since substituting any configuration into the argument locations still results in a parenthesis boundary form. The compound form is called a pseudo-operator because it is just one particular boundary form; accepting it as a specific operator would require accepting all boundary forms as specific operators.

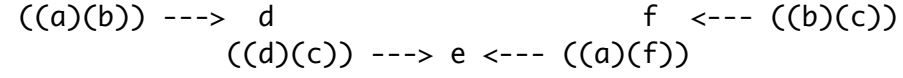
Any space within an arbitrary boundary form can accommodate any number of arguments, making the arity of the operator quite ambiguous. Thus, the compound operator ((.)(.)) does not even represent a class of structures. The general class would include structures with an arbitrary number of space two levels deep, ((.)(.)(.)...). Examples include

((.)) ((.)(.)(.)) ((.)(.)(.)(.)(.)(.))

Continuing anyway:

5.2.1 Associativity (a ◊ b) ◊ c = a ◊ (b ◊ c)

Diagrammatically,



Again, substitution achieves a simpler display format:

((((a)(b)))(c)) = ((a)(((b)(c))))
------- -------
d f

INVOLUTION converts this equation of boundary associativity into an identity equation.

((((a)(b)))(c)) = ((a)(((b)(c))))
((a)(b) (c)) = ((a) (b)(c)) inv

((.)(.)) has been shown to be associative. This is slightly better than the case of SHARING because the partitioning by INVOLUTION is more natural.

5.2.2 Identity $a \lozenge i = i \lozenge a = a$

The boundary form of the identity constraint for the compound operator ((.)(.)) is

```
((a)( i )) = (( i )(a)) = a                transcribe
```

The identity is ( ), which can be shown by substitution

```
((a)(( ))) = (((  ))(a)) = a               subst i = ( )
((a)     ) = (     (a)) = a                 inv
  a        =        a   = a                 inv
```

Thus far, the compound binary form ((.)(.)) is a monoid when ( ) is the identity. This applies to the entire class of such forms.

5.2.3 Inverse $a \lozenge a^{-1} = a^{-1} \lozenge a = i$

In boundary form, (a) is the inverse of a.

```
((a)((a))) = (((a))(a))   =   ( )          transcribe
((a)  a  ) = (  a  (a))   =   ( )          inv
(( )  a  ) = (  a  ( ))   =   ( )          per
           =            =!=  ( )          occ
```

The expected contradiction; the boundary form ((.)(.)) fails the inverse test, and does not form a group.

In sum, there are no configurations of boundaries in boundary algebra that provide a group function $\lozenge$. This is of interest because:

1) Since Boolean algebra operators do form groups, boundary and Boolean algebras are not isomorphic, and

2) Boundary algebra provides a new approach to computation that is at variance with group theoretic principles.

There are a number of other difficulties that arise in comparing the group structure of Boolean and boundary systems. For example, AND and OR form two Boolean groups that combine to form a Boolean ring. A ring requires two associative, commutative and distributive operators. There are not two operators available in boundary algebra.

The Boolean system is a distributed, complemented lattice, with AND as meet, and OR as join. AND and OR are dual. The "Boolean dual" of SHARING is another form of SHARING, nested two levels deep, ((.)(.)). This form is not an operator any more than any other configuration of boundaries is an operator. It is a compound form consisting of three BOUNDINGs and one SHARING. The form could be called the dual of SHARING, but it is dual only when a Boolean interpretation is imposed upon it. This is another form of the circular argument that explicitly constructs boundary forms to match Boolean properties, and then identifies boundary algebra as a Boolean algebra.

An idea about how group theory might be extended to accommodate boundary algebra is considered next.

5.3 A New Kind of Identity

For SHARING, the monoid identity is Void, which technically does not exist. The definition of inverse, required to form a group, results in a contradiction when the monoid identity is used. For boundary algebra in general:

> The monoid identity is the inverse of the group identity.

Basically, group theory itself requires enlargement, to include something like an "inverse-identity", i^{-1}:

Identity for SHARING:

$a \lozenge i = i \lozenge a = a$		Identity
$a \quad i = i \quad a = a$		$\lozenge$ = SHARING
$a \qquad = \qquad a = a$		i =

Inverse for SHARING:

$a \lozenge a^{-1} = a^{-1} \lozenge a = i^{-1}$		Inverse
$a \ (a) = (a) \ a = i^{-1}$		$\lozenge$ = SHARING
$\quad ( ) = ( ) \qquad = ( )$		$i^{-1} = (i) = ( )$

That is, the identity element, i, defined by the identity equations is the inverse of the identity element, i^{-1}, defined by the inverse equations.

6 SUMMARY

Boundary algebra is an application of boundary mathematics, a collection of new formal tools and techniques for computing using spatial forms rather than tokens recorded on a line. Both the syntax and semantics of boundary mathematics are incommensurable with conventional mathematical systems. For example, the structural properties addressed by group theory, including the definition of a function, are properties only of linear, string-based notations.

This paper emphasizes the substantial differences between boundary and Boolean algebras, and highlights the one-to-many mapping from boundary to Boolean systems as a primary differentiator. The most significant conceptual leap required to understand boundary mathematics is to accept that nothing, Void, can be used semantically without engaging a syntactic representation.

The mathematics of spatial forms is pattern-based, but not strictly functional or relational. Syntactic variants of spatial notations are generated by topological and geometric transformation rather than by token substitution and rearrangement. Non-representation permits global transformation through the transparency property of PERVASION, while void-equivalent forms can be used freely as reduction catalysts without concern about syntactic or semantic interactions.

6.1 Table of Non-Correspondence

	Boolean	Boundary	Difference
symbols	atomic	spatial	linear vs topological
constants	{0, 1}	{ ( ) }	two vs one
unary operator	NOT	BOUNDING	delimited collection
binary operator	OR, AND	SHARING	not a function, not binary
arity	specific	any	no concept of argument
mapping	functional	relational	one-to-many
ordering	implication	bounding	spatially explicit
computation	rearrange	delete	void-equivalence
semigroup	associative	no concept	boundary structure only
monoid	identity, i	void	existence
group	inverse	i^{-1}	new structure needed
Abelian group	commutative	no concept	no spatial metric

6.2 List of Differences

Boundary algebra is formally more succinct than Boolean algebra due to a one-to-many mapping that maintains equivalent expressive power.

Boundary algebra does not include these conventional ideas:

Cardinality	--	Boolean and boundary systems are idempotent.
Associativity	--	Space does not support grouping.
Commutativity	--	Space does not support metric structure.
Arity	--	Space does not support counting of contents.
Functions	--	Void-substitution does not support functions.

Boundary algebra introduces these concepts that are not present in conventional systems:

Semantic void/single constant --
: Void is everywhere; boundaries distinguish their contents. Nothing more.

Inside and outside of forms --
: Containment makes a partial order relation explicit.

Spatial (non-linear) notation --
: Typography becomes topology.

Boundary algebra computation includes these techniques that, for the most part, are not present in conventional transformational approaches:

Object/operator unification --
: Patterns are both objects and operators.

Semipermeable boundaries/operational transparency --
: Boundaries are barriers to their contents, transparent to their context.

Void-equivalence --
: Void-equivalent structure is semantically irrelevant and semantically inert.

Virtual computation --
: Void-equivalent forms catalyze transformation.

The above features come as a package; they characterize boundary algebra, and they are not independent. The features provide powerful algebraic and computational techniques that have no parallel in conventional mathematics.

7 REFERENCES

B. Banaschewski (1977) On G. Spencer Brown's Laws of Form. Notre Dame Journal of Formal Logic, 18: 507-9.

W. Bricken and P.C. Nelson (1986) Pure LISP as a Network of Systems. Proc. Second Annual Kansas Conference: Knowledge-based Software Development, Kansas State U.

W. Bricken and E. Gullichsen (1989) An Introduction to Boundary Logic with the Losp Deductive Engine. Future Computing Systems 2(4): 1-77.

W. Bricken (1992) Spatial Representation of Elementary Algebra. 1992 IEEE Workshop on Visual Languages, IEEE Press, 56-62.

W. Bricken (Naoki Kobayashi and Sueki Matsumura trans.) (1993) Extended Abstract: A Formal Foundation for Cyberspace. Intercommunication #3.

W. Bricken (1995) Distinction Networks. in I. Wachsmuth, C.R. Rollinger & W. Brauer (eds.) KI-95: Advances in Artificial Intelligence., Springer, 35-48.

P. Cull and W. Frank (1979) Flaws of Form. Int. J. General Systems, 5: 201-11.

W.E. Gould (1977) Review of Laws of Form. Journal of Symbolic Logic, 42: 317-318.

N.S.K. Hellerstein (1997) Delta: A Paradox Logic. World Scientific.

E.V. Huntington (1933) New Sets of Independent Postulates for the Algebra of Logic. Transactions AMS 35: 274-304.

J. James (1993) A Calculus of Number Based on Spatial Forms. M.S.E. Thesis, University of Washington. http://www.lawsofform.org/collection.html

L.H. Kauffman (1985) Notes from the Laws of Form Seminar Chicago 1975. Proceedings First Annual Conference/Workshop on Sign and Space, Santa Cruz.

L.H. Kauffman (1987) Self-reference and recursive forms. J. Social and Biological Structures, 10: 53-72.

L.H. Kauffman (1990) Robbins Algebra. Proc. 20th International Symposium on Multiple Valued Logic, IEEE Press, 54-60.

L.H. Kauffman (1995) Arithmetic in the Form. Cybernetics and Systems 26: 1-57.

L.H. Kauffman (2001) The Mathematics of Charles Sanders Peirce. Cybernetics and Human Knowing, 8(1-2): 79-110.

L.J. Kohout and V. Pinkava (1980) The Algebraic Structure of the Spencer-Brown and Varela Calculi. Int. J. General Systems, 6: 155-71.

W. McCune (1996) http://www.mcs.anl.gov/home/mccune/ar/robbins

P.G. Meguire (2003) Discovering Boundary Algebra: A Simple Notation for Boolean Algebra and the Truth Functors. Int. J. General Systems 32: 25-87.

C.S. Peirce (1933) Collected Papers -- IV Chapter 3 -- Existential Graphs, 4.397 - 4.417. C. Hartshorne and P. Weiss (eds), Harvard U. Press.

R.A. Orchard (1975) On the Laws of Form. Int. J. General Systems 2: 99-106.

D.D. Roberts (1973) The Existential Graphs of Charles S. Peirce. Mouton.

D.G. Schwartz (1981) Isomorphisms of Spencer-Brown's Laws of Form and Varela's Calculus for Self-reference. Int. J. General Systems 6: 239-255.

R.G. Shoup (1993) A Complex Logic for Computation with Simple Interpretations for Physics. PhysComp'92 Workshop on Physics and Computation, IEEE Press.

G. Spencer-Brown (1969) Laws of Form. George Allen and Unwin Ltd.

F.J. Varela (1975) A Calculus for Self-Reference. Int. J. General Systems, 2: 5-24.

F.J. Varela and J.A. Goguen (1978) The Arithmetic of Closure. J. Cybernetics, 8: 291-324.

F.J. Varela (1979) Principles of Biological Autonomy. Elsevier North Holland.

L.L. Whyte (1972) Review of Laws of Form. British Journal of Philosophy of Science, 23: 291-292.