BOUNDARY LOGIC FROM THE BEGINNING

William Bricken

September 2001, updated October 2002

Boundary Logic from the Beginning

What is the simplest mathematics? In finding a simpler foundation for mathematics we also find a variety of simpler and stronger computational tools, algorithms, and architectures.

Guide to Reading

The contents are structured for different levels of reading detail.

> Sections delineate origins, concepts, and applications to logic and computation.
> Small subsections describe the new concepts and tools.
> NEW IDEAS summarize the important conceptual content in one-liners.
> SIDEBARS add historical context, depth commentary, and examples.

SECTION 1. DISCOVERING ICONIC LOGIC

Symbolic logic uses symbols to express the concepts of logic. *Symbols* are chosen arbitrarily, their structure bears no resemblance to their meaning. Iconic logic uses icons, graphs and diagrams to express the concepts of logic. An *icon* resembles what it refers to. Thus, iconic logic is a visual presentation of the meaning of logic.

Boundary logic (BL) is one type of iconic logic. Others are Venn diagrams, Karnaugh maps, and circuit schematics. BL is an iconic depiction of the mathematical concept of enclosure, interpreted as the logical operation `IMPLIES`.

Like any mathematics, BL is a way of looking at problems, a way of thinking and seeing. As a technique, it may seem unfamiliar and incomprehensible at first. After some practice, it will seem second nature. Since BL is a simpler system of logic, there is much less to learn.

Our strategy is to begin with the most simple mathematical idea, and then to make uncontroversial observations until we have observed enough to construct a model of computation. The goal is to learn a simpler way of thinking about logic, and thus to develop simpler computational algorithms and architectures.

What we discover is that logic exhibits its geometric nature through an algebra of enclosures.

1.1 ORIGINS

The American logician and philosopher Charles S. Peirce first conceived of and published results in boundary logic in 1898. Peirce called his work *Existential Graphs*, and considered it to be among his greatest accomplishments. Existential graphs were sufficiently foreign an idea that both historians and biographers have omitted substantive portions of this work, considering them to be senile ravings.

The next major contribution to BL was George Spencer-Brown's 1969 book *Laws of Form*, a mathematics text that caused considerable excitement at the time by introducing void-based logic within an algebraic framework. Influential logicians who studied *Laws of Form* concluded that it was "just another representation of Boolean algebra", one without significantly interesting properties. They failed to appreciate the fundamental nature of the concepts, and again BL was suppressed as "without content".

Recognizing that boundary logic lacked a convincing application, William Bricken developed computational implementations of BL from 1978 through 2002.

1.2 BEGINNING

Mathematics uses tokens to represent both concrete objects and abstract concepts. The first thing we do in communicating mathematically is to clear a space for writing and drawing tokens, called the *representational space*. The cleared space rests upon a physical substrate. The representational space itself, abstractly, is void. As yet it possesses no defining characteristics. Outside of representational space, on the other side of the boundary that identifies and bounds the emptiness, is physical space.

This then is the simplest idea, the simplest beginning. We start with a representational space, initially empty and bounded. An empty blackboard, a blank sheet of paper, a smooth stretch of sand. Prior to marking, the representational space is void-space, or more simply, *void*. An empty page, for example, is void of marks.

Void-space has absolutely no structure and no properties. When the first mark is brought into existence by being scribed upon void-space, it takes on the property of existing. Prior to that, nothing is supported, nothing exists. Void-space precedes the mark as an ineffable, without distinction of any kind.

A mathematical concept with no defining properties is most difficult for those trained in conventional logic to understand. Many have begun: "Let the void-space be labeled by the letter Phi." This elementary error, naming that which does not support a name, is the primary reason why some contemporary mathematicians have failed to understand BL.

Technically, the only way to indicate void-space is indirectly. The simplest method is to identify an enclosure that bounds void-space. Thus, the boundary of a page, that transition between physical and representational space, is an indicator that we have set aside void-space. We can refer to the indicator as an indirect route to that which does not support direct reference.

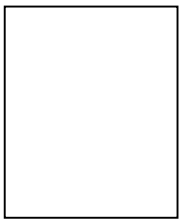
Another technique is to identify a void-equivalent form, one that exists yet possesses the same intent as void-space, that of representational irrelevance. In logic, for example, double negation is void-equivalent, since `NOT NOT A = A`.

1.3 MARKS

Next, in order to communicate something, we mark the representational space in some way. We cancel, or negate, void-space by occupying it with a *mark*.

The mark can mean anything we choose. What that mark means is defined by our intention when marking. At this point, the intention is to indicate simply what is observable. Aside from the physical substrate (board, paper, sand), thus far we can observe the physical presence of an enclosure, the boundary that separates physical-space from void-space:

physical-space
is out here



Without introducing new content, the initial mark must refer to the boundary between the two spaces. We could make a meaningless scribble and identify it with the boundary of void-space. This type of mark is a symbol. We could draw a picture of what we see. This type of mark is an icon. Above, the iconic rectangle bears a resemblance to what it represents while the symbolic word "physical-space" is an encoded shape bearing no resemblance to physical space.

NEW IDEA The icon of void-space is the mark.

The iconic mark possesses the same structural properties with respect to representational space, as does the physical page it represents with respect to physical space. Specifically, there is an outside, there is void inside, and there is an observable boundary between the two. As an icon, the mark illustrates an enclosure. An enclosure is defined by a containing boundary. The iconic rectangle represents our intention to mark, and is the product of that intention. Indirectly, the existence of a mark declares the existence of a sentience making the mark.

SIDEBAR: Mathematical Symbols and Icons

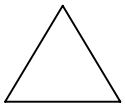
Pure mathematics can be considered to be the study of token structures, with no attempt to relate these structures to any usage, interpretation, or application. Symbolic mathematics is the formal study of symbol strings. Iconic mathematics is the formal study of iconic forms.

A *symbol* is an association between an actual object or concept and an arbitrary token. We take advantage of this arbitrary choice by using symbols composed of letters and punctuation-marks, tokens that fit nicely into lines, thus providing display convenience.

An i*con* is an association between an actual object or concept and a token that resembles that object or concept. Thus, part of the meaning of an icon is its specific shape.

Icons are preferable to symbols since icons convey more information. The Arabic numeral 2, for instance, lacks the visual cardinality of the Roman numeral II. A photograph is of a greater informational intensity than is a verbal description.

Some mathematical forms are inherently iconic, such as an equilateral triangle.

equilateral triangle 

A formal icon can be translated into linear symbols, such as the words "equilateral triangle", or symbols derived from labeling the parts of the triangle: a = b = c, <AB = <AC = <BC. The icon, however, carries more information, since it illustrates as well as describes.

Symbolic description is easily standardized. More importantly, we have developed calculi to manipulate symbols (look-up tables, substitution of equals, logical deduction, term rewriting, theorem proving), permitting automated computation. Calculating with icons is thought to be difficult and clumsy. However, several iconic calculi have recently been studied extensively; these include fractals, cellular automata, particle diffusion, and knot theory.

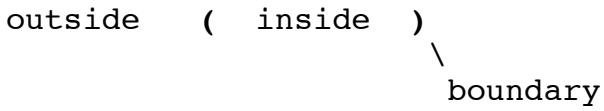
The triangle icon above refers to a mathematical object that cannot exist in physical space except as an approximation. We readily accept representational forms, such as photographs, of physical objects that do exist. A drawing can, in contrast, resemble an object that does not physically exist. In the case of mathematical objects, we are willing to believe in the existence of non-physical objects. The ideal equilateral triangle is assumed to exist, it is easily visualized. In BL, the icon of logical implication bears a resemblance to, rather than symbolizes, what we know as implication, or more generally, rationality. The difficulty is that we have no experience with what implication looks like. We are thus forced to break new conceptual ground not only with regard to iconic logic, but also with regard to pictorial thought.

The dilemma we face is coming to understand how the familiar conversational logic embedded in language is equivalent to iconic forms of boundaries. We must also come to understand why the iconic form is so much more efficient than the symbolic form.

For example, culturally we accept that the symbol NOT means, but does not look like, the concept of logical negation. The icon ( ) also means logical negation, yet the mark is not chosen arbitrarily, it is intended to look like the concept. Specifically, an enclosure, through containment, separates its contents from everything else. The boundary icon *is* a separation, as well as referring to separation. The mark acts on its inside by asserting that the inside is NOT outside.

1.4 DELIMITERS

It is convenient to express enclosures in typographical form. This can be achieved by using a textual delimiter (paired textual tokens such as parentheses, brackets, braces, and the like). Visually:



Above, there is no "left-outside" or "right-outside", since outside is a two dimensional construct that spills over the boundaries of the linear text string.

Parentheses used iconically as enclosures are called *parens.*

`( )` ***PARENS***

Parens forms are constructed by nesting and concatenating parens. A *well-formed parens* configuration (WFP) is one that has matching left and right delimiting tokens. The set of WFPs represent ways in which parens can contain each other. On the page, WFPs form nested enclosures that do not intersect. The structural constraints of containment and enclosure are same as the typographical constraints of balanced parentheses. The vocabulary of the parens language is all possible WFPs.

SIDEBAR: Typographical Conventions of Parens Notation

Several notational conventions help to adapt reading habits to the iconic forms of BL.

Parens Notation

Parens, `( )`: When a boundary is represented as a typographical string, the typographical enclosure manifests as a matched pair of symbols. However the two "sides" of the parenthesis are aspects of a single boundary object, a single enclosure. Habitually we read the left parenthesis as "open" and the right as "close", but this imposition of linear unfoldment is an inaccurate characterization. The only distinction an enclosure makes is that of inside and outside; the dual textual symbols are an artifact of typography.

Equality, `=`: The equal sign asserts that the two representational spaces that it distinguishes contain forms with the same intention, or meaning. It asserts that forms on each side can be validly transformed into the same form. Equality also asserts valid substitutions; we can always substitute the form on one side for the form on the other side.

Curly braces, `{ }`: Curly braces are iconic variables, a notation for arbitrary configurations of enclosures, including none or many. Symbolic notation uses a textual label, such as `x` or `y`, to identify a variable value. These labels do not have an inside, and thus cannot

stand in place of a configuration of enclosures. In order to distinguish between different curly brace forms, they must also contain a label. Thus {A} is a configuration of enclosures labeled A. "A" is not a form or a configuration, it is an identifier of a particular brace configuration.

When an equation containing an iconic variable does not use the variable's inner space, the curly braces will be dropped; the more conventional single letter will then be used to identify iconic variables. Iconic variables stand in place of single forms, collections of forms sharing a space, or the absence of forms.

Conventions to Aid Reading

Void, <void>: The label <void> is a typographical convenience so that the eye trained in symbolic equations does not find the absence of a symbol strange. The symbol <void> is simply a reminder to our eyes, it has no meaning within the context of parens tokens. The angle brackets are a notational convention that emphasizes that the token within identifies an object in a different representational system.

<void> =

Square brackets, [ ]: Like <void>, square brackets are solely a visual aide, used to emphasize particular sets of parens. A parens, ( ), and a square bracket, [ ], mean exactly the same thing. Square brackets are *highlighted* parens. Brackets draw the eye to a particular parens that, for the purposes of exposition, is currently being emphasized. For example, the following form emphasizes the outer enclosure: [( )]

Typographical space, : It is often helpful to visually emphasize the erasure or deletion of parens within a single form. This can be accomplished by leaving empty the typographical space formerly occupied. A sequence of transformations can be illustrated by showing a sequence of changing forms, each differing from the prior by an erasure:

```
( (( )) (( )( )) )          original form
(       (( )( )) )          cross
(       (( )   ) )          call
(              )             cross
```

1.5 UNARY LOGIC

Binary logic uses two symbols {0, 1} as computational objects. Boundary logic uses one icon. That icon has already been constructed, it is the mark ( ), the icon of void-space. Boundary logic is *unary* since it uses only one token. That token is an enclosure that differentiates two spaces.

Both binary and boundary logic express the same logical concepts of implication and deduction, however each uses a completely different set of tools and representations.

NEW IDEA Binary logic distinguishes pairs of values.
Boundary logic distinguishes pairs of spaces.

The additional expressive power in BL, which permits abandoning one of the binary tokens, comes from the two dimensional, iconic nature of the mark. Marks support two spaces, an outside and an inside. The outside space contains the mark, it is the marked (and thus not void) space. The mark both names and identifies the space that contains it. The inside space is void, preserving a portion of void-space for further representational activity.

As we shall see, the mechanism thus far, void-space and its iconic boundary (drawn self-referentially upon itself) is sufficient, when observed, to generate the entirety of logic.

1.6 ENCLOSURE AS OBJECT, ENCLOSURE AS PROCESS

One way to characterize the properties of a mark is to consider it to be an enclosure. A mark is an empty enclosure. The fundamental property, that a mark differentiates two spaces, can be expressed in the language of enclosures:

NEW IDEA An enclosure contains some forms and excludes others.

Paper, blackboards, web pages, TVs, projection screens and computer monitors are all two dimensional enclosures of symbolic and iconic information. Rooms, cars, boxes and bottles are also enclosures, they are three dimensional containers of physical objects.

Importantly, *enclosure* is both an object and a process. As an object, enclosure is represented by a mark as seen from outside. As a process, enclosure actively operates on its contents by separating them from the outside.

NEW IDEA Enclosure is the only type of object and the only type of process.

Enclosure serves as both object and operator, undermining the distinction between the two. Conventionally, operators {`AND`, `OR`, `NOT`} apply to objects {`TRUE`, `FALSE`}. The independence of the two domains is enforced by unique naming and unique structural positioning. In BL, the single object type `( )` and the single process `( )` are indistinguishable. Furthermore, the structure of a WFP contains both syntactic and semantic information. As an object, the structure represents logic; as a process, the structure specifies how to compute the value of the logical object.

The distinction between object and operator depends on whether the enclosure is viewed from the outside or from the inside. Viewed from its outside, we see an enclosing mark. Viewed from its inside, we see that we are enclosed. How we interpret the intent of an enclosure is a convention of reading, not of representation. The choice of object or operator is one of perspective, not definition.

The containment relation is mathematically unique. It is not an ordering relation, since ordering requires that either A contains B or B contains A. In the parens form ((A)(B)), neither is the case. Containment is not a partial ordering either. A partial order is reflexive. Enclosures are not reflexive since they do not contain themselves.

1.7 REPLICATION AS DIVERSITY

The initial act of representation is to mark void-space. Marking again allows us to construct configurations of enclosures using *replicas* of marks. Configurations of marks are called *forms*. Each act of replication comes with a choice, since we can place that replica on the inside or the outside of a given enclosure.

We can visualize this by using the parens icon:

```
( ) ( )        Two enclosures outside each other.

( ( ) )        One enclosure inside another, a double-container.
```

Forms within a space share that space, creating *sharings*.

```
( ) ( )        elementary sharing
```

Forms within an enclosure nest inside, creating *nests*.

```
( ( ) )        elementary nest
```

Replication naturally leads to counting and to the integers, which is too complex for our current purposes. Replication also provides a variety of forms that can potentially express semantic diversity. The entire set of possible forms constitutes the parens language. Some examples follow:

```
(((((  )))))

(  )  (  )  (  )  (  )

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

(((  ))  (((  ))(  ))  (((  )(  )(  )((  ))))))
```

SECTION 2. CONCEPTS OF BOUNDARY LOGIC

Since the new concepts of BL have been developed from a simple basis, they are easy to understand, and support a rich variety of physical analogies. The challenge of BL is to understand why logic is iconic.

Boundary logic introduces three fundamentally new concepts: void-space, enclosure, and transparency.

> ***NEW IDEA*** Void-space is absence of all properties.
> Enclosure defines an inside, an outside, and a boundary.
> Enclosures are transparent to replicas.

These concepts are not usually associated with our ideas of what logic is. Nevertheless, boundary logic can be interpreted as, or applied to, conventional logic. Although using one mathematics to describe another does not anchor either in reality, it does establish that the two are functionally equivalent.

In terms of traditional logical concepts, BL uses

- mark and void as `TRUE` and `FALSE`.
  A mark means `TRUE`, the absence of a mark means `FALSE`

- enclosure as logical implication.
  The inside of an enclosure `IMPLIES` the outside.

- transparency as the process of logical deduction.
  If a form is on the outside, it need not be on the inside.

2.1 THE FREEDOM OF VOID-SPACE

Unlike geometric space, void-space has no associated metric or curvature. There are no points or distances in void-space. Size and position are irrelevant concepts. Void-space supports only observation of existence.

Space without structure or properties is fundamentally different than space with properties. In particular, void-space does not support the mathematical concepts of ordering (commutativity), grouping (associativity), or counting (cardinality). Representational space does support emptiness, variarity, and independence.

Representational space can contain as many forms and replicas as we choose, including none. Although empty space is not a functional operator, it supports the existence of any arbitrary number of boundary forms (variarity). Emptiness refers to the absence of a form.

In general, forms sharing space do not interact with each other, since representational space has no structure upon which to build a concept of interaction. The currency of space is existence; further, when used for iconic logic, only boundaries exist in this space. Thus, forms can be addressed and transformed in parallel, without interference with each other. Enclosures sharing space are functionally orthogonal.

> ***NEW IDEA*** Space may contain any finite number of forms, including none. Forms sharing space do not interact.

SIDEBAR: Symbolic Representation

Voice and text both unfold in a sequential stream over time. This structure requires symbolic codes that do not mimic their intention. Symbolic codes enforce a division between representation and reality, between mind and body, due to the arbitrary nature of their forms. Thus, words and actions apply to quite different domains.

Words are linear and symbolic. Symbolic representation has been accepted by the mathematical community at least since Descartes. It is currently an absolute standard for mathematical communication. Rigor is better served through a unified encodement; the symbiotic union of logical proof and mathematical notation, homogenized through digital computation, is a basis that assures manageable formal processes.

The language of concepts is not necessarily linear or symbolic, as we know from art, theater, film, music and architecture. Expression in these fields is iconic; intention is conveyed through vision and experience rather than through trained memory.

Symbolic representation, like any encodement, imposes characteristic limitations on the form of expression, in exchange for a commonly learnable code. Teaching the linear symbolic code (reading, writing and arithmetic) is the primary activity throughout educational institutions.

The limiting presumptions built into symbolic mathematics require explicit permissions to overcome. These limits include:

- arity (counting):
  The number of objects that an operation can address is specific and fixed.

- associativity (grouping):
  The grouping of multiple objects imposed by specific arities is important.

- commutativity (ordering):
  The order of applying an operation to multiple objects is important.

Iconic formal systems express concepts in two and three dimensions, and can be shown to be equivalent to linear, one dimensional representations. Iconic representations do not naturally impose structural rules for grouping, ordering, and counting.

2.2 ENCLOSURE AS PERSPECTIVE

Spaces are environments, void-space is the environment of tokens. Conventionally we consider an environment to have no visible boundaries and no limit on contents. Type-written pages are an example. The empty page is void-space. It has a physical boundary that we learn not to see while reading. The world is outside, while text and pictures are inside this explicit boundary. We have, of course, many elaborate conventions of reading and writing that maintain the illusion that typographical space is unlimited.

To a large extent, our physical bodies impose the distinction between inside and outside. When the visible boundary is as wide as our peripheral vision, we consider ourselves inside. When the boundary is within our foveal vision, we are definitely outside.

It is our personal *perspective* that determines whether we consider something to be an object or an environment. We are outside of objects, inside of environments. Shifting perspective from inside to outside converts environments to objects. For example, the atmosphere of Earth is the encompassing environment we dwell in; when seen from a space station, however, it is a beautiful object. Shifting perspective from outside to inside converts objects into environments. For example, an automobile seen from the street is a fast moving object; while inside driving it, it is a travelling environment.

NEW IDEA Personal perspective is a shift operator.

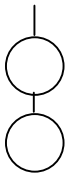
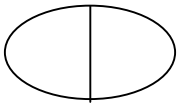
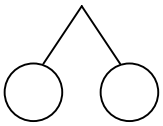
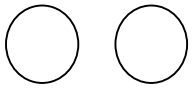
We use our perspective to shift between object and environment, between content and context, between data and process, between container and containment, between outside and inside. Perspective as operator provides unique computational properties. Immediately, the reader, as a mathematician, is involved in a form. The objective/subjective barrier is breached. The intention of the mark depends upon our focus of attention.

BL involves the reader by permitting multiple interpretations without inconsistency. Common words, of course, also possess the property of multiple interpretations. Conventional mathematics, in contrast, has sought to eliminate multiple interpretation of a single symbolic form by enforcing unique labels and unique definitions. For boundary forms, the choice of treating an enclosure as an object or as an operation is solely a matter of computational convenience.

SIDEBAR: Containment, Connection and Contact

Enclosures define a containment relation between two spaces, and can be represented by well-formed configurations of delimiters. The containment relation can also be expressed as a directed acyclic graph. Each graph node is a specific enclosure. Arcs entering a node identify forms inside that node. Arcs exiting a node identify the outer enclosure that encloses that node. In this form, containment becomes connection.

A graph representation can be converted into a map representation by making nodes into territories and arcs into common borders. In this form, containment becomes sharing a common boundary. Thus, in a different iconic representations, enclosure (parens) is also connection (graphs) and contact (maps). The following are identical forms expressed in these three different iconic notations:

	parens	***graphs***	***maps***
nest	(( ))		
sharing	( ) ( )		

2.3 PERVASION AND TRANSPARENCY

Representational space is *pervasive*, it is everywhere throughout every form. Enclosures form hierarchies of spaces. Any outer space pervades, or permeates, all inner spaces. However, a space can never escape its containing boundary; pervasion does not apply outwardly. Pervasion establishes the equivalence of void-space across all marks or boundaries. That is, void-space underlies all forms. In boundary logic, forms are pervasive as well. Forms pervade all inner spaces, they exist within all inner boundaries.

Forms that are outside a given enclosure can be replicated in any space inside that enclosure without changing the logical intention. Replicas that are pervaded (by an identical form) can be deleted without changing the logical intention. Enclosures are *transparent*, or permeable, to replicas. What is outside can be replicated and then freely put inside. When a replica is inside, it can freely be deleted.

Transparency does not extend outwards. Forms on the inside that are not replicas of some outside form cannot escape their enclosure. If this were the case, then enclosures would not distinguish spaces since replicas could be placed anywhere.

Some illustrations of transparency follow, the first and second lines of each example are semantically equivalent:

```
  A A
= A
```

```
  A (A)
= A ( )

  A (B (A C D))
= A (B (  C D))

  A (B (A (C (B (A (C D E)))))
= A (B (  (C (  (  (  D E)))))
```

2.4 REDUCING DIVERSITY

Adding transformation rules to our set of observations builds a calculus. The following boundary equations define an algebraic calculus of logical evaluation.

In the Boolean domain, configurations of constants and operators can be evaluated, reducing the configuration to a single value, `0` or `1`. Similarly, forms can be reduced to either a single mark, or to void, to nothing at all. The two rules of reduction, Calling and Crossing, mimic our intuitive understanding of how empty space works.

NEW IDEA Replicas are void-equivalent.

`( ) ( ) = ( )` ***CALLING***

This equation says that two marks sharing space is equivalent to one mark in that space. Read right to left, *Calling* gives explicit permission to replicate marks. Read from left to right, Calling declares that replication means nothing and changes no intentions.

Thus Calling both recognizes the possibility of duplication, and stops that possibility from generating a cardinal sequence. In a sense, with the rule of Calling, replication cannot generate the integers because two is equal to one. Without this rule, boundary logic turns into boundary numbers.

We can interpret Calling in other ways as well. For instance, two empty pages provide the same markability as one. We freely introduce another page when one is full, with the intention of extending the original territory. Two pages means the same as one page.

NEW IDEA Double-enclosures are void-equivalent.

`(( )) = <void>` ***CROSSING***

Crossing is the void-equivalent rule, it identifies forms that are irrelevant. Interpreted for the marked page, Crossing gives permission to erase the mark.

The original change was to mark void-space. We then introduced replication of marks, and somewhat rescinded it with Calling. Crossing rescinds the original marking, returning forms to void. Without this rule, boundary logic turns into boundary sets.

SECTION 3. BOUNDARY LOGIC

Our objective to observe the simple has accumulated a number of tools and concepts. But what have we observed? We began by clearing a portion of physical space so as to be able to make a mark. Without introducing content, we observed the boundary between physical and representational space, and recorded it as an iconic mark. We then gave ourselves permission to reuse that which we already had, the ability to mark. The process of choosing whether to place a new mark inside or outside existing marks, continued indefinitely, naturally yields the set of all possible yes/no decision trees, a structure that also characterizes all possible finite logical formulae.

The abstract concepts of boundary mathematics therefore have a direct application to formal logic. Calling and Crossing define the mechanism of logical evaluation. When supplemented with variables, they define the mechanism of logical deduction.

The symbols of elementary logic are linguistic in origin. Conceptually, they carry the limitations of a linear notation. It is possible to convert textual logic into iconic logic. Both are logic; one is familiar, linear, symbolic, duality-based and clumsy, the other is completely unfamiliar, planar, iconic, void-based and efficient.

We might say that formal logic is iconic as well as symbolic. Typographical symbols disguise a higher dimensional process. Throughout history, logic has been equated with the functioning of the rational mind. Iconic logic provides an illustration of this process, a picture of rational thought.

SIDEBAR: Logic, Algebra, and Geometry

Geometry is the study of spatial structures and relationships. Euclid (c. 300BC) introduced geometrical thinking; his axioms addressed the properties of points and lines in planar space. Descartes (1637) showed that by placing a metric on space, the famous Cartesian coordinates, we could convert points and lines into algebraic equations. Algebra itself evolved from the Renaissance as the study of numerical forms, first applied to accounting, later rigorously explored as polynomial equations.

In contrast, logic came from language. It is embedded into our thoughts and expressions as the familiar `AND`, `OR`, `IF`, and `NOT`, and has been since antiquity. The origins of logic are linear and sequential, conforming to the structure of words and speech. Aristotle (c. 340BC) introduced the first symbolic system for logic. This system of syllogisms was carried through the Dark Ages by rote memorization of textual forms. Boole (1854) presented the first formal system of logic. Only in the early 1900s was logic formalized into a complete and consistent symbolic system, independent of both spoken language and semantic content.

During the development of formal logic, both iconic and algebraic forms were explored (Venn, Kempe, Peirce, Frege, Russell and Whitehead). Russell's textual, symbolic form based on implication was eventually adopted throughout the logic community, the iconic forms being

considered too limited and clumsy. This completely separated logic from geometry, a choice that was cemented with the advent of sequential digital computers.

During the 1970s, different computational approaches to logic were developed, including the union of logic and algebra in equational logic. Historically, logicians and algebraists formed separate, relatively non-communicating communities. Equational logic is not favored by either community since it is an undesirable hybrid that confounds the essential concepts of each discipline. Geometers too are a separate community, residing predominantly in the branch of Physics that studies the nature of space. Manipulation of geometrical configurations and proof of geometrical theorems is strictly a symbolic, logical exposition, as are mathematical proofs. Although spatial visualization and intuition may lead one to geometrical understanding, this understanding is converted into textual logic during the course of mathematical communication.

Thus today logic, algebra and geometry are seen as separate disciplines. Boundary logic rejoins them into a unified system that uses iconic forms in equations to compute logical outcomes.

3.1 ENCLOSURE AS LOGIC

We begin with the elementary truth values {`TRUE`, `FALSE`}. We have only enclosure, delineated space, and replication as boundary tools.

Let a mark be taken as the truth value `TRUE`.

NEW IDEA The concept `TRUE` is a mark.

```
( ) = TRUE
```

The reading of the mark is from the outside, using it as an object. If a mark exists then it is equivalent to the symbol `TRUE`. We have confounded truth with existence.

Marking void-space changes it from void to marked. We could say that the mark negates the void. Since `TRUE` negates `FALSE`, void is equivalent to `FALSE`:

NEW IDEA The concept `FALSE` is void-equivalent.

```
<void> = FALSE
```

Void-space itself absorbs that which is `FALSE`. In BL, the concept `FALSE` simply does not need to exist. We attribute no properties other than irrelevance to `FALSE`, since void-space has no properties.

We could begin by taking the void-equivalence of `FALSE` as fundamental. In this case, marking void-space negates its emptiness, changing the intention of the space from `FALSE` to `TRUE`. Thus, the mark indicates `TRUE`.

NEW IDEA An enclosure negates its contents.

```
(A) = NOT A
```

An enclosure negates the freedom of its contents. The reading of the mark is from the inside, using it as an operator. Enclosures act to constrain, they are opaque from the inside.

The double-enclosure, `(( ))`, iconically represents the cancellation, or erasure, of a mark. The inner enclosure is a mark, the outer enclosure negates, or changes, that mark by returning it to unmarked space.

```
<void> =          FALSE

  ( )  =     NOT FALSE =     TRUE

 (( )) = NOT NOT FALSE = NOT TRUE  = FALSE = <void>
```

SIDEBAR: Table of Correspondence

Logical Operators	***Parens Forms***
FALSE	<void>
TRUE	( )
NOT a	(a)
a OR b	a b
a AND b	((a)(b))
a IMPLIES b	(a) b
IF a THEN b ELSE c	(((a) b)(a c))
a XOR b	((a) b)(a (b))
a IFF b	(a b)((a)(b))

Here is an example of the encoding process, translating logic into parens.

```
(NOT (a AND (b OR c)))

          (b OR c)   =>          b c
     (a AND (b OR c))  =>   ((a) (b c))
(NOT (a AND (b OR c))) =>  (((a) (b c)))  =>  (a) (b c)
```

Here is an example of the decoding process, translating parens into logic. A consequence of BL being formally simpler than symbolic logic is that decoding provides many different alternatives. That is, the mapping between parens and logic is *one-to-many*. One parens form stands in place of many different logic expressions. How a parens form is read as logic is at the discretion of the reader. For example, the form

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

can be read in a variety of ways, including

```
a AND b
(a AND b) OR FALSE
(NOT ((NOT A) OR (NOT b)))
(NOT (a IMPLIES (NOT b)))
((NOT a) OR (NOT b)) IMPLIES FALSE
```

3.2 LOGICAL OPERATORS

Containers have two asymmetric sides. We can use this to define the concept of logical implication, the operator `IMPLIES`.

NEW IDEA The inside implies the outside.

```
(A) B  =  A IMPLIES B
```

A boundary is an hypothesis, postulating inside and concluding outside. We can test this interpretation by observing elementary forms. Below the enclosure used as implication is highlighted. In the first row, the implication that void implies void is the same as the form for `TRUE`. Similarly, we can complete the truth table for material implication:

```
[    ]      = ( )          FALSE IMPLIES FALSE = TRUE
[    ] ( ) = ( )          FALSE IMPLIES TRUE  = TRUE
[( )] ( ) = ( )          TRUE  IMPLIES TRUE  = TRUE
[( )]      =              TRUE  IMPLIES FALSE = FALSE
```

These rules are visually consistent. What we take as our initial equation is the identity of marks. Anything sharing a space with `( )` is also void-equivalent, this is confirmed by two equations. `(( ))` is equal to void, this is confirmed by two different equations. Calling and Crossing each appear as a special case.

Reading parens forms in different ways leads to the other logical operators. Sharing space, the iconic concept of independence, is the logical concept of disjunction, the operator `OR`:

NEW IDEA Sharing space is disjunction.

```
A  B  =  A OR B
```

The truth table for space sharing is that of logical OR:

```
          =                        FALSE OR FALSE = FALSE
      (  ) = (  )                  FALSE OR TRUE  = TRUE
(  )      = (  )                   TRUE  OR FALSE = TRUE
(  ) (  ) = (  )                   TRUE  OR TRUE  = TRUE
```

We start with the indistinguishability of void-space and the identity of marks. Only Calling is necessary to verify the consistency of the mapping. Since OR is equated with void-space, disjunction in BL has no properties. Instead, forms sharing space have the property of independence.

SIDEBAR: Equations and Inference

We have been taught to think of mathematical concepts as they are applied to numbers. These same ideas apply to enclosures. Like the arithmetic and algebra of numbers, BL uses the familiar properties of equality. Specifically,

- Substitution
  We can substitute equals for equals.

- Replacement
  We can replace one form with what it is equal to throughout any equation.

- Transformation
  If a series of substitutions or replacements converts one form into another, the two forms are equal.

- Convergence
  If a series of substitutions or replacements converts two forms into the same form, the original two forms are equal.

- Functional invariance
  Applying a function to equal forms maintains equality.

In contrast, logic uses implication to derive conclusions from premises. Implication and thus logical inference is one-directional; implication is asymmetrical. When A IMPLIES B, we know that when A is TRUE, so is B. If we know only that B is TRUE, we know nothing about A. Thus, inference progresses through the accumulation of truths from assumed premises. We can never discard known facts and rules, since they may provide the premise for yet another conclusion.

Algebra proceeds in both directions; equality is symmetrical. When we know that A EQUALS B, we can use either to represent our current state of knowledge. Only one need be preserved since

nothing is lost by discarding the equivalent expression. Thus, algebra proceeds through transformation of expressions.

BL combines algebra and logic. Forms can be processed using equality or using inference, the choice is a matter of computational convenience.

SIDEBAR: Logical Equality

Boundary systems can use void descriptively. A valid boundary equation is

```
<void> = <void>
```

More properly in the iconic language,

```
=
```

The above equation asserts that void is equal to void. The type of equality here is more elementary than identity, it is better considered to be indistinguishability.

The elementary equation of identity is

```
( ) = ( )
```

Here, equality asserts that a mark is indistinguishable from a mark.

Thus, the iconic intention of equality is to allow us to observe forms that are not different, while also asserting that marks are what they appear to be. Forms on each side of an equal sign have indistinguishable intentions.

Boolean Equality

The principle of Boolean equality connects algebra and implication. It states that two expressions are equal if each implies the other. Symbolically:

```
A = B      means      ((A->B) & (B->A)) => TRUE
```

This same principle can be expressed in BL:

```
A = B      means      [[[A] B] [[B] A]] => ( )
```

We can now logically prove whether or not two forms are equal. Consider first the invariance of void-space. Let `A = <void>` and `B = <void>`:

```
  =        means      [[[ ]  ] [[ ]  ]] => ( )
```

The proof requires two parallel applications of Crossing, leaving the mark of Truth:

```
[ [[ ]  ] [[ ]  ] ]
[                 ]          cross (twice)
```

Consider the identity of marks:

```
( ) = ( )    means     [ [[( )] ( )] [[( )] ( )] ] => ( )
```

This parens form reduces to mark via four applications of Crossing:

```
[ [[( )] ( )] [[( )] ( )] ]
[ [      ( )] [      ( )] ]          cross (twice)
[                         ]          cross (twice)
```

3.3 DEDUCTION

Deduction is the technique of logical proof. Logical deduction is defined inductively by a *recursive iconic equation,* with a base and an inductive case. In the following equations, `{B A}` means any form `A` anywhere within form `{B}`. By any form, we mean any WFP, including `<void>` and collections of forms. By anywhere, we mean in any subspace regardless of nesting depth or shared contents.

NEW IDEA The enclosure of a mark is void-equivalent.

```
(( ) A) = <void>                    OCCLUSION   base case
```

Occlusion is the rule of evaluation. It identifies specific irrelevant configurations: those enclosures that contain a mark are irrelevant. This is the base case of the definition of boundary deduction.

NEW IDEA Replicas are void-equivalent.

```
A {B A} = A {B}                     PERVASION   inductive case
```

Pervasion is the rule of deduction. It identifies irrelevant forms: replicas of forms are irrelevant regardless of their location. This is the inductive case of the definition of boundary deduction.

Pervasion is constrained by outer enclosures, just as tokens are constrained by the page boundary. Calling, the rule of replication, initiates a replica in the same space as the original. Pervasion gives permission for replicas to be placed anywhere within any forms sharing space with the original.

Occlusion and Pervasion are the algebraic reduction rules for BL. Both proceed via void-substitution. Occlusion is a hybrid rule, it encompasses all the functionality of Crossing and

Calling, while more efficiently deleting forms that contain a mark. Pervasion is purely algebraic, depending upon identification of arbitrary replicas.

SIDEBAR: Examples of Boundary Deduction

In the proofs below, four different BL transformation rules are used to change a symbolic problem statement into a conclusion. These are:

`(( ) A) = <void>`	***OCCLUSION***
`( ) A = ( )`	***DOMINION***
`((A)) = A`	***INVOLUTION***
`A {B A} = A {B}`	***PERVASION***, insert <=> extract

Boolean Equivalence

We can determine whether two algebraic forms are equal by placing them within the form of Boolean equivalence and reducing the resulting form. A proof progresses by the deletion of nested replicas, using the algebraic rule of Pervasion. For example, suppose we want to know if the Involution equation were `TRUE`:

`a = ((a))` ***INVOLUTION***

First convert the equation into its logical format. Substituting,

let `A = a, B = ((a))`

`[ [[A] B ] [[ B ] A] ]`	form of equivalence
`[ [[a] ((a))] [[((a))] a] ]`	substitution

The proof proceeds by transparency. Inner forms that replicate outer forms are deleted.

`[ [[a] ((a))] [[((a))] a] ]`	
`[ [[a] ( )] [[((a))] a] ]`	extract (a)
`[ [[a] ( )] [[(( ))] a] ]`	extract a
`[ [[a] ( )] [[ ] a] ]`	cross
`[ ]`	occlude (twice)

Reading a proof in reverse provides a constructive path to the desired equation. This is particularly helpful within computational implementations of BL.

Modus Ponens

The axiom of *modus ponens* is the primary deductive rule for conventional logic. Here it is shown to be a consequence of BL transformations.

```
((A AND (A IMPLIES B)) IMPLIES B)          MODUS PONENS

     (((A) ((A) B))) B                     transcription

     (( (A) ((A) B) ))  B
     (( (A) (    B) ))  B                  extract (A)
     (( (A) (     ) ))  B                  extract B
     (                 )  B                occlude
     (                 )                   dominion
```

Absorption

This theorem of conventional logic requires Pervasion to be used in a constructive direction, inserting selected forms as well as extracting them. The proof reduces the left-hand-side of the equation to the right-hand-side.

```
(A AND (A OR B)) = A                       ABSORPTION

     ((A)(A B)) = A                        transcription

     ( (A) (    A B) ) = A
     ( (A) ((A) A B) ) = A                 insert (A)
     ( (A) (( ) A B) ) = A                 extract A
     ( (A)           ) = A                 occlude
        A              = A                 involution
```

Deductive Challenge

The final example is a complex logical deduction problem. These types of problem are usually formulated in words and then translated into logical symbols. The translation process requires an intimate knowledge of the structures and intentions of language. The symbolic proof using conventional Natural Deduction is difficult. The BL proof is algorithmically simple.

Given the three premises, prove that the conclusion is `TRUE`:

Premise 1: If he is lying, then (if we can't find the gun, then he'll get away).

Premise 2: If he gets away, then
(if he is drunk or not careful, then we can find the gun).

Premise 3: It is not the case that (if he has a car, then we can find the gun).

Conclusion: It is not the case that he is both lying and drunk.

Encode the propositions as symbols:

L = he is lying
G = we can find the gun
A = he will get away
D = he is drunk
C = he is careful
H = he has a car

```
P1:       IF L THEN (IF (NOT G) A)
P2:       IF A THEN (IF (D OR NOT C) THEN G)
P3:       NOT (IF H THEN G)
C:        NOT (L AND D)
```

Encode the symbolic logic as parens and simplify each premise:

```
P1:       (L) ((G)) A     => (L) G A
P2:       (A) (D (C)) G
P3:       ((H) G)
C:        (((L)(D)))      => (L)(D)
```

Join all premises and conclusions into one form, using the logical structure:

```
if (P1 and P2 and P3) then C
```

The parens form of this is:

```
[ [[P1] [P2] [P3]] ] C  =>  [P1][P2][P3] C
```

Substituting the forms of the premises and conclusion yields:

```
[(L) G A] [(A) (D (C)) G] [((H) G)] (L)(D)
```

The BL proof now follows:

```
[(L) G A] [(A) (D (C)) G] [((H) G)] (L)(D)
[(L) G A] [(A) (D (C)) G]   (H) G   (L)(D)        involution
[      A] [(A) (D (C))  ]   (H) G   (L)(D)        extract G (L)
[      A] [    (D (C))  ]   (H) G   (L)(D)        extract (A)
[      A]       D (C)       (H) G   (L)(D)        involution
[      A]       D (C)       (H) G   (L)( )        extract D
                                        ( )        dominion
```

SECTION 4. BOUNDARY COMPUTATION

Mathematical techniques change over time, they evolve and refine and extend what was previously known. Formal logic, for instance, is about 100 years old. Prior to that, logic was largely memorization of particular rules, rules that derived from careful observation of language. Formal logic provided a basis that supported the evolution of computation, since it provided an understanding of the Boolean domain based on algorithmic rules applied to symbols rather than on linguistic patterns and meanings.

BL suggests some fundamental changes in the way we view and implement computation. In particular,

- Presence and absence is sufficient to express binary logic,
  rather than {0, 1} or {TRUE, FALSE} or {on, off} or {HIGH, LOW}.
- Inside and outside is sufficient to express logical operations,
  rather than {AND, OR, NOT, IMPLIES}.
- Void-substitution is sufficient to implement binary evaluation.
- Transparency is sufficient to implement logical deduction.
- Masking void-equivalent forms is potentially more efficient than
  propagation of clocked signals through logic networks.

4.1 LOGIC AND CIRCUITRY

Silicon computation, at its basis, is the presence or absence of a flow through a network of dynamic obstructions. We interpret the flow and the absence of flow of electric current through a transistor, called {HIGH, LOW}, as Boolean values {1, 0}.

All computing we do today is based on Boolean circuitry, this has not changed since the inception of computing about 70 years ago. A very early design decision was to aggregate small transistor networks into a more abstract design concept, logic gates composed into networks. *Logic networks* then became the currency of circuit design, as they still are today. Transistor networks all trigger simultaneously when a clock permits a serge of current to identify, relatively instantly, the available flow paths. Logic design assures that wires that have a HIGH value when the circuit is pulsed are uniquely associated with variables that would be TRUE should the logic function corresponding to the circuit be evaluated.

BL realigns logic with the physical basis of circuitry, the presence or absence of a flow. Rather than using logic networks that are transparent to electrical pulses, BL circuitry can be implemented using latching networks, circuit components that are not transparent to flows. When a flow reaches a latch, the latch either disconnects from the network it is in, or it does

nothing (i.e. is transparent). The result of a computation, the output, is determined by whether the exit latch is left open or closed.

NEW IDEA BL circuitry propagates disconnections.

SIDEBAR: A Small Digital Circuit

A BL circuit can be represented by a parens form with labels as inputs. For example, the following BL circuit has three inputs {a, b, c} and computes the two-out-of-three majority function.

```
((a b)(a c)(b c))                    2/3 majority
```

Inputs are either mark (connect) or void (disconnect). Assume the following input vector:

```
a = HIGH = mark   = ( )
b = LOW  = <void> =
c = HIGH = mark   = ( )
```

Substituting these bindings yields a parens form without variables

```
(((  )  )((  )(  ))(  (  )))
```

Evaluating this form consists of deleting all enclosures that contain a mark. E.g.:

```
(((  )  )((  )(  ))(  (  )))        input
(                        )          occlude (thrice), output
```

Three parallel deletions, or void-substitutions, reduce in one step the original form to a single mark, the mark of Truth. That is, the function returns HIGH since at least two inputs are HIGH.

4.2 ENCLOSURE AS COMPUTATION

Parens map directly onto conventional logic networks, providing an interpretation of parens forms as circuits. Each enclosure can be taken to be a logical NOR with a variable number of inputs.

NEW IDEA The parens language represents the set of possible computations.

Parens map to both logic structure and circuit structure, providing a common iconic language for the two. The observation is thus far restricted to circuits of a special type, combinational circuits with no loops and no registers.

For a circuit to evaluate, all inputs must be bound to either a `HIGH` or a `LOW` signal. Execution, or evaluation, of a circuit is arithmetic, using logic values as abstractions of signal levels. Parens without variables identify the possible ways circuits can be evaluated; they describe circuit activity.

Circuitry is designed without regard to specific input values. Circuit design is algebraic, using logic equations and variables as abstractions of transistor networks and physical pins. Parens with variables identify the possible ways circuits can be designed; they describe circuit functionality.

Occlusion is the only evaluation rule necessary for determining the output of a circuit given a set of inputs. This suggests a body of hardware implementations that use void-equivalence as the sole computational principle.

Pervasion is the only transformation rule necessary for optimization of circuit elements. Due to the transparency of enclosures, Pervasion allows direct transformation of circuit components that are at a distance from each other. Transparency provides new tools for circuit optimization, tools that address the outstanding problem of minimization of *multilevel* logic networks.

The single boundary operator permits homogeneous models of computation at both the software and the hardware levels. The characteristic confounding of object and operator in boundary logic manifests as a confounding of logic gates and connective wires, permitting a fluid partitioning and allocation of logic and routing resources. And the simple structure of forms provides simple structural models of circuits that yield quickly to hierarchical abstraction and bit-width vectorization from the bottom-up. Most importantly, since parens forms are just nested lists, EDA logic synthesis, technology mapping, and placement and routing can be implemented solely with sorting and pattern-matching algorithms, both well understood and non-exponential.

4.3 SOFTWARE DESIGN TOOLS

The Iconic Logic Optimizing Compiler (ILOC) is a suite of three hierarchical software tools calling upon a collection of hundreds of boundary logic algorithms. At the core, the *Boundary Logic Engine* provides Boolean minimization of multilevel logic networks. The *Circuit Graph Engine* manipulates circuitry-specific transformations on structure, such as technology mapping. The outer layer of functionality, the *Application Interface*, admits and generates EDIF specifications, provides batch-mode processing, test-vector verification, statistical analysis, and interfaces to EDA and other interactive systems.

ILOC can efficiently generate functionally invariant circuits across a wide diversity of specified configurations. It's primary function is circuit reconfiguration to meet specified parameters. Since these tools are based in the formal system BL, all transformations are formally correct, maintaining the functional intent of the circuit at all times.

SIDEBAR: Iconic Logic Hardware Design Tools

General Boolean Capabilities

- evaluate logic expressions in given binding environment
- minimize logic expressions algebraically
- identify tautologies
- determine satisfiability and generate counterexamples
- perform partial function evaluation
- minimize variable references
- identify abstract symmetries
- identify necessary sequential and parallel components of computation
- standardize logic representations
- verify equivalence of structurally different forms
- Boolean factoring
- Boolean pattern abstraction
- Boolean constraint reasoning
- case analysis

Boolean Tools Applied to Circuit Optimization

- network decomposition
- redundancy removal and minimization
- don't care analysis and minimization
- critical path modification
- network decomposition and transformation
- specification of maximum fanin and fanout
- structure abstraction
- library mapping
- structure sharing
- test vector generation and BDD generation
- technology mapping
- construct abstractions and hierarchical library elements
- convert to asynchronous, object-oriented graph-reduction model

Circuit Reconfiguration Capabilities

Once a locally minimal logical form of a circuit is generated, the form can be manipulated to achieve specified design constraints.

- deepening (distribution),
- structure sharing (coalesce),
- trading-off connectivity for depth (factorization)
- matching to structural templates (technology mapping)
- partitioning for place and route (hierarchical abstraction)
- retiming (register relocation)

Controllable Design Parameters

- low-redundancy (3 levels: literals, forms, distribution of forms)
- CNF/DNF, SoP/PoS
- implicate normal form
- generalized nor
- N-input look-up tables
- specific gate sets: {nor, not}, {and, or, not}, {and, or, not, mux, xor}
- specified fanin and fanout
- structure sharing (increased fanout, decreased area)
- maximal structure sharing (disassemble and reconstruct)
- depth/wiring tradeoffs (critical path reduction)
- low-level abstraction (xor and if-then-else/mux units)
- high-level structural abstraction (specific to functionality)
- specific structural partitioning (designated library modules)
- parens form (no partitioning)
- occlusion arrays
- partial function evaluation (some bound inputs)

EDA Specific Capabilities

- parse and minimize EDIF circuit descriptions
- optimize logic for combinational and sequential multilevel circuits
- detect false paths and minimize reconvergent fanout
- remove don’t care conditions
- reduce delay along critical path
- reduce gate count or area
- exchange timing for wiring
- optimize structure sharing in multilevel circuits
- specify fanin and fanout limits
- identify equivalent circuits and pin-point differences
- generate circuits to meet multiple design objectives
- identify xor and mux patterns and generate hierarchical decompositions
- generate test vectors
- identify abstract patterns and construct library elements
- generate technology mapping for xor, mux, n-LUT and other architectures
- simulate functionality with bit-streams

SECTION 5. CONCLUSION

Boundary logic expresses conventional logic and computation with entirely new concepts. Half of each logical dual (`TRUE/FALSE`, `AND/OR`, `ALL/EXISTS`) is void-equivalent, and thus does not exist to participate in computation. Enclosure is the only concept necessary to express logical techniques. Logical evaluation consists of deleting marked enclosures. Logical deduction consists of finding replicate forms in nests.

These new concepts are minimal. They reduce the complexity of representing both logical ideas and computational circuits, leading to efficient new logical operations, deductive processes, and software and hardware architectures for digital computation.

As well, BL provides a new economy of thought. Truth exists. Falsity is non-existent, it is simply not meaningful for rational thought.

Rationality and logical deduction arise naturally and inevitably as soon as we agree to enter formal icons onto a representational space. Entering replicas does not increase information (Calling). Changing and changing back, conceptual oscillation, does not increase information (Crossing).

From these premises and from observation, it is a small step to consider our rational thought processes as beginning in void-space. We add marks, choosing to put them inside or outside existing marks. We evaluate by forgetting enclosures of marks. We deduce by forgetting nested replicates.

Conventional logic has always harbored a contradiction. It is said to be "how we think rationally", yet symbolic logic is notoriously difficult. Simple problems of inference are known to confuse the majority of people. Perhaps the most penetrating idea introduced by the iconic approach to deduction is that what we know as rationality is not the formal symbolic manipulation of complex and challenging logical formula. Rather it is a simple and inevitable consequence of making marks in empty space.

SIDEBAR: Summary of New Ideas in Boundary Logic

The icon of void-space is the mark.
Binary logic distinguishes pairs of values. Boundary logic distinguishes pairs of spaces.
An enclosure contains some forms and excludes others.
Enclosure is the only type of object and the only type of process.
Void-space is absence of all properties.
Enclosure defines an inside, an outside, and a container.
Enclosures are transparent to replicas.
Space may contain any finite number of forms, including none.
Forms sharing space do not interact.
Personal perspective is a shift operator.

Summary of Equations

Calling: Replicas are void-equivalent. `( ) ( ) = ( )`

Crossing: Double-enclosures are void-equivalent. `(( )) = <void>`

Occlusion: The enclosure of a mark is void-equivalent `(( ) A) = <void>`

Dominion: Forms sharing a space with a mark are void-equivalent. `( ) A = ( )`

Involution: Double-enclosures are void-equivalent. `((A)) = A`

Pervasion: Replicas are void-equivalent. `A {B A} = A {B}`

Logic:

The concept TRUE is a mark.	`( )`	`= TRUE`
The concept FALSE is void-equivalent.	`<void>`	`= FALSE`
An enclosure negates its contents.	`(A)`	`= NOT A`
The inside implies the outside.	`(A) B`	`= A IMPLIES B`
Sharing space is disjunction.	`A B`	`= A OR B`

SIDEBAR: Glossary

double-enclosure: The elementary nest, `(( ))`.

enclosure: A simple representation that encloses space, creating a distinction between the inside and the outside. Enclosure is both object and operator.

form: Nests and sharings of enclosures.

icon: A representation that bears a resemblance to what it means.

mark: An empty enclosure, `( )`.

nest: Enclosures within enclosures. The elementary

parens: A typographical notation that uses parentheses to represent enclosures.

perspective: A reading operation that chooses between object and process.

replica: A copy or a duplicate of a form

representational space: That space set aside from physical space for the purpose of recording tokens.

sharing: Enclosures occupying the same space. Elementary sharing, `( ) ( )`

symbol: An arbitrary encodement of an intention.

transparent: Void-equivalent replicas can exist anywhere within a form.

unary: Using one single operator or token.

variarity: Spaces may contain an unspecified number of forms.

void: The absence of symbols and icons. Void has absolutely no mathematical properties.

void-equivalent: An existent form that has the intention assigned to void, usually irrelevance.

void-space: Same as void.

void-substitution: Deleting a void-equivalent form.

well-formed parens: A balanced parenthesis structure.

WFP: Well-formed parens.