Boundary Logic

Advances in knowledge must necessarily appear to be unintelligible *before* their discovery and simple or obvious *after* their discovery.
-- Arthur Fischell

Challenge

Computation and logic (Boolean algebra) are universally built on binary representations.

```
0     1                    True  False             Yes   No
```

Is there a simpler approach? Can logic be expressed in a *unary* notation?

Boundary Mathematics

The use of delimiting tokens, or *containers*, as both constants and functions.

Pure math example: *Common boundaries cancel*.

$$\boxed{\boxed{}} \quad = \quad \texttt{<void>}$$

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

Concepts

Boundary token	an enclosure
Representational Space	the bounded space

The Simplest Virtual World

<this space is intentionally left blank>

<the above contradicts itself>

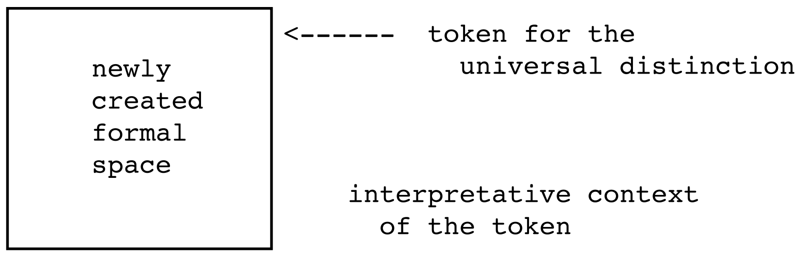
Two Voids

Absolute void:	that which cannot be referred to without contradiction
Relative void:	emptiness enclosed within a boundary

Constructing a Distinction

A **Universal Distinction** is first boundary we agree upon. In forming a universal distinction, we construct three things simultaneously:

a formal space (inside)
an interpretative context (outside)
a token representing the distinction (boundary)

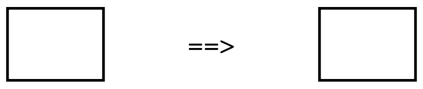

newly created formal space
<------ token for the universal distinction
interpretative context of the token

Calling

Focus your attention on the outside,
where you see the mark (the usual viewing point).

Call the boundary that you see a “symbol”.

To call is to maintain perspective.



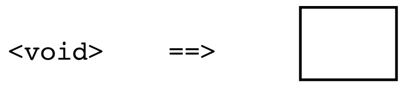
Calling is the rule of **invariance**. It is also is the rule of **naming**. Thus the relationship between an object and its name is invariant.

Crossing

Focus your attention on the inside,
where you cannot see a mark (there is no mark inside).

Cross the boundary to the outside. Now you can see a mark.

To cross is to change perspective.



Crossing is the rule of **variance**. It is also a process of changing.

The Arithmetic of Boundaries

```
        CALLING             ( ) ( )  =  ( )

        CROSSING            ( ( ) )  =
```

Moving to Algebra

The ground, or carrier set, of boundary logic is one token { ( ) } and one absence of a token. If an equation holds for all ground values, it holds in general. Using this, we can construct algebraic truths from the cases of the arithmetic:

```
        DOMINION            INVOLUTION          PERVASION

        ( ) ( ) = ( )       (( ( ) )) = ( )     ( ( ) ) ( ) = ( )
        ( )     = ( )       ((     )) =         (     )     = ( )
thus
        ( )  A  = ( )       ((  A  )) =  A      (  A  )  A  = ( )
```

Boundary Logic Algebraic Axioms

The transformation axioms of boundary logic:

Dominion (the halting condition, when to stop)

```
( )  A  =  ( )                   REIFY   <==>  ABSORB
```

Involution (double negation, how to remove excess boundaries)

```
((A))  =  A                      ENFOLD  <==>  CLARIFY
```

Pervasion (how to remove excess variable mentions)

```
A (A B)  =  A (B)                INSERT  <==>  EXTRACT
```

Each axiom suggests a definite reduction strategy:
erase irrelevant structure
to convert the left side of the equation to the right side.

Algebra provides the useful tool of **substitution independence**. Any transform can be applied at any time and at any place in the expression without changing the value of the expression. Thus, all transformation paths do not change the value of an expression. It doesn't matter how you get to a simpler expression (an answer). Some paths may be longer and less efficient, but all lead to equivalent results.

Boundary Logic

Boundary logic uses a **spatial representation** of the logical connectives. Since CALLING provides an object-centered interpretation, and CROSSING provides a process-centered interpretation of the same mark, boundary forms can be evaluated using either an algebraic (match and substitute) process or a functional (input converted to output) process.

Representation of logic and proof in spatial boundaries is new, and quite unfamiliar. Boundary logic is not based on language or on typographical strings, nor is it based on sequential steps. Boundary techniques are inherently **parallel and positional**. The meaning, or interpretation, of a boundary form depends on where the observer is situated. From the outside, boundaries are objects. From the inside, you cross a boundary to get to the outside; boundaries then are processes. This dramatically different approach (that is, permitting the observer to be an operator in the system) does not change the logical consequences or the deductive validity of a logical process.

Spatial representations have built-in associativity and commutativity. The base case is no representation at all, that is, **the void has meaning** in boundary logic. Simplification of logical expressions occurs by **erasure of irrelevancies** rather than by accumulation of facts.

Boundary Logic Representation

logic	*boundary*	*comments*
False	`<void>`	no representation. Note: `(( ))` = <void>
True	`(  )`	the empty boundary
A	`A`	objects are labeled by tokens
not A	`(A)`	
A or B	`A  B`	Disjunction is sharing the same space
A and B	`((A)(B))`	
if A then B	`(A) B`	Implication is separation by a boundary
A iff B	`(A B)((A)(B))`	

In the above map from conventional logic to boundaries, the many textual forms of logical connectives condense into one boundary form. Note that the parens, `(  )`, is a linear, or one-dimensional, representation of a boundary. Circles and spheres are expressions of boundaries in higher dimensional representations.

Multiple Readings of the Same Form

A single expression in the simpler notation of boundary logic can express (infinitely) many different forms in a more complex notation.

For example:

`( (A) (B) )`	A and B
	(not ((not A) or (not B)))
	(not (A implies (not B)))
	((not A) or (not B)) implies False

Comparative Axiomatic Basis

An **axiomatic basis** is a minimal set of transformations from which all other transforms can be derived. The basis of conventional logic:

```
P -> (Q -> P)                                      isTrue
((P -> False) -> False) -> P                       isTrue
(P -> (Q -> R)) -> ((P -> Q) -> (P -> R))          isTrue
```

Transcribing the conventional basis of logic to boundary logic:

```
(P) (Q) P                          =  ( )
(((P))) P                          =  ( )
((P) (Q) R) ((P) Q) (P) R          =  ( )
```

The basis of boundary logic is (mathematically) beautiful:

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

Boundary Logic Examples of Proof

To Prove:	*Transcribe*	*Steps*

`A implies A`	`(A) A`	`( ) A`
		`( )`

`not not A = A`	`((A)) = A`	`A = A`

`A or A = A`	`A A = A`	`A ((A)) = A`
		`A (( )) = A`
		`A       = A`

`A and B =`
`not (not A or not B))`
	`((A)(B)) =`
	`((A)(B))`
		`identity`

```
(and (A implies B) A) implies B     (( (A) ((A) B) ))  B

                      (( (A) ((A) B) ))  B
                         (A) ((A) B)     B          involution
                         (A) ((A)  )     B          pervasion of B
                         (A) (     )     B          pervasion of (A)
                             (     )                dominion
```

A Constructive Proof

SUBSUME A and (A or B) = A

```
                      ( (A) (    A B) )  =  A       transcribe
                      ( (A) ((A) A B) )  =  A       insert (A)
                      ( (A) (( ) A B) )  =  A       extract A
                      ( (A) (( )    ) )  =  A       absorb A B
                      ( (A)           )  =  A       clarify
                         A               =  A       identity, qed.
```

Truth Table Example in Boundary Logic

Example: `if (P and Q) then (R iff (not S))`

Transcribe into boundaries:

```
(P  and  Q)         ((P) (Q))
(R iff (not S))     (R (S)) ((R)((S)))  =  (R (S)) ((R) S)
if... then...       (((P) (Q))) (R (S)) ((R) S) = (P) (Q) (R (S)) ((R) S)
```

The expression is True whenever Dominion applies. Erasing variables sets them to False:

```
When P is False, it is erased:  ( ) (Q) (R (S)) ((R) S)   =  ( )      dominion
When Q is False:                (P) ( ) (R (S)) ((R) S)   =  ( )      dominion
```

Note that the form `(X (Y)) (Y (X))` is True when `X` is not the same as `Y`. Substituting:

```
(P) (Q) ( (( ))) (( ) ( ))  =  (P) (Q)  ( )        =  ( )
```

and when R is True and S is False

```
(P) (Q) (( ) ( )) ( (( )))  =  (P) (Q)        ( )  =  ( )
```

These four cases identify all the True forms of the expression.

Conversely, the expression is False only when everything vanishes, that is, when

(P is True) and (Q is True) and ((R is True, S is free) or (S is True, R is free))

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

Natural Deduction Example in Boundary Logic

Premise 1:	If A then (if (not P) C)	the Fruit problem
Premise 2:	If C then (if (O or not K) then P)	
Premise 3:	Not (if B then P)	
Conclusion:	Not (A and O)	

Encode the logical connectives as boundaries, and simplify:

P1:	`(A) ((P)) C`	=	`(A) P C`	involution
P2:	`(C) (O (K)) P`			
P3:	`( (B) P )`			
C:	`( ((A) (O)) )`	=	`(A) (O)`	involution

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

```
        (P1 and P2 and P3) ->  C
```

The proof structure of "conjunction of premises imply the conclusion" as boundaries:

```
        ( ((P1) (P2) (P3)) ) C  =>  (P1) (P2) (P3)  C    involution
```

Substituting the forms of the premises and conclusion, and reducing:

```
( (A) P C )  ( (C) (O (K)) P )  ( ((B) P) )  (A) (O)
( (A) P C )  ( (C) (O (K)) P )     (B) P     (A) (O)    involution
( (A)   C )  ( (C) (O (K))   )     (B) P     (A) (O)    pervasion of P
(       C )  ( (C) (O (K))   )     (B) P     (A) (O)    pervasion of (A)
(       C )  (     (O (K))   )     (B) P     (A) (O)    pervasion of (C)
(       C )         O (K)          (B) P     (A) (O)    involution
(       C )         O (K)          (B) P     (A) ( )    pervasion of O
                                                 ( )    dominion
```

Interpret the final form: `(   )` = True

Boundary Quantification

All x. P(x)	`(x) Px`	x -> Px isTrue
Exists x. P(x)	`( (x) (Px) )`	x and Px isTrue

Quantifier relations:

All x. P(x)	iff	(not (Exists x. (not P(x))))	`(x) Px`	`= (( (x) ((Px)) ))`
All x. (not P(x))	iff	(not (Exists x. P(x)))	`(x)(Px)`	`= (( (x) (Px) ))`
(not (All x. P(x)))	iff	Exists x. (not P(x))	`((x) Px )`	`= ( (x) ((Px)) )`
(not (All x. (not P(x))))	iff	Exists x. P(x)	`((x)(Px))`	`= ( (x) (Px) )`