EQUALITY IS NOT FREE

William Bricken

November 2005

CONTENTS

Our conceptualization of mathematical expressions, definitions, and proofs is formulated in the language of logic, using AND and NOT and IMPLIES and IF-AND-ONLY-IF. This same language maps to boundary algebra, so that the way we describe and address problems *logically* can also be formulated as the structural transformation of algebraic boundary forms.

Boundary algebra provides a different way of thinking about deduction and rationality. The language of boundary algebra consists only of SHARING and BOUNDING and EQUALS.

In the sequel, we use boundary algebra tools to analyze and to deconstruct the structure of logic itself, with an emphasis on the relationship between logical connectives and algebraic EQUALS.

BOUNDARY ALGEBRA

Boundary algebra is based upon three simple initial equations. This set of rules, called OPI herein, are sufficient to characterize any Boolean algebra.

(A ( )) =	OCCLUSION
A {A B} = A {B}	PERVASION
((A)) = A	INVOLUTION

These particular three equations constrain the interpretation of this use of boundary mathematics to Boolean algebra. In particular, the inclusion of PERVASION as a rule anchors boundary algebra to logic. A boundary algebra that includes PERVASION is called boundary logic.

An understanding of the formal basis of boundary logic would not necessarily effect the computational use of boundary mathematics tools. It would, however, shed light on the relationship between boundary and Boolean algebras, and thus help to define the nature of logic and the nature of mathematics itself.

BOUNDARY LOGIC

The three boundary logic rules are a toolkit for structural manipulation of parens forms, especially for reducing them to minimality. Since they map onto elementary logic, they are also a toolkit for mathematical deduction and proof. But are these rules independent? Are all three necessary? Is there an even more elegant way to formulate logic?

The following discussion is structural rather than semantic; structural displays show definitions and transformations toward minimality. Reference to logical meaning ceases at the formal description of the problem, prior to transcription into parens forms, and is resurrected only when the answer is available at exit.

Thus, parts of the technical narration herein are in the language of boundary mathematics; the novice might consider this language as a series of machine instructions to achieve a goal. There is also an undeniable need to describe boundary logic processes in terms of the original transcription language, that is, in terms of conventional logic, deduction and proof. In almost all cases, transcription is immediately available before, during, and after boundary logic transformations. Back-transcription becomes unavailable during boundary logic processes that have no mapping to conventional formal techniques.

The narration provides reasons for making specific transformations. From the structural view, the goal is always to delete void-equivalent forms. From the narrative view, the transformation sequences usually service goals that are external to the structural steps, such as the choice to define a lemma so that a subsequent proof is easier to describe.

In particular, we will prune boundary logic even further, by showing that INVOLUTION is a consequence of the other two rules. This permits the formulation of conventional deductive proof techniques as a single recursive equation, with OCCLUSION as the base case, and PERVASION as the inductive case.

CONSIDER EQUALITY

The use of algebraic equations and the EQUALS sign in the rules of boundary logic carries with it a substantive set of built-in assumptions. Algebra in general provides the familiar transformation techniques of replacement and substitution. Equations define constraints on variables in expressions. Algebraic representation places requirements on the use of variables; for instance, naming must be unique and names can stand in place of arbitrary expressions.

The EQUALS sign identifies equivalence classes of expressions. It has the properties of identity, transitivity, and commutativity which support the validity of the algebraic transformation techniques. Importantly, EQUALS is a Boolean connective, the value of an assertion of equality is either True or False.

Herein, we use IFF to identify both Boolean EQUIVALENCE and symmetric logical consequence, and the EQUALS sign to identify algebraic boundary structures with the same value. The letters X and Y are used solely for the definition of equality throughout the rest of the paper.

Some variations of the EQUALS sign are also used:

=?=	might be equal, to be proved
=!=	not equal
==>	can be reduced to, by applying rules
-->	equal by transcription between two languages
=def=	equal by definition

The primary goal is to make the connection between logical IFF and algebraic EQUALS explicit. In particular, what logical theorems and mechanisms are inherently incorporated in an equation? The sequel demonstrates that the EQUALS sign used within the rules of boundary logic already incorporates some of the rules that it also defines. The particular initials of boundary logic (and by mapping, the rules of inference and deduction) are not axiomatic structural equations, they are in part consequences of any algebra that uses EQUALS.

IF AND ONLY IF

The language of logic is conventionally cast with an implicational vocabulary; for an interpretation as logic, equality is IF-AND-ONLY-IF (IFF). IFF is freely used as a conventional proof technique by constructing two required cases, the necessary (IF) condition and the sufficient (AND ONLY IF) condition.

The structure of IFF is dictated by the other Boolean connectives, but unlike the other collectives, there are two minimal structural varieties of IFF. Any use of the EQUALS sign incorporates a representational choice between the structural varieties of IFF.

```
(X IMPLIES Y) AND (Y IMPLIES X)          IFF-one

(NOT (X OR Y)) OR (X AND Y)              IFF-two
```

That IFF has two minimal forms is not a priori apparent in the conventional language of logic, because the metric of operator complexity is confounded in that language. Not only does each operator have a specific inverse (negation of the operator), it also has duals such as AND and OR that must be assigned equivalent metrics.

In logic, IMPLIES is historically the most fundamental connective. IFF-one has two units of implicational complexity, plus whatever AND costs. IFF-two, which is restricted to the connectives of Boolean algebra, has one unit negation and three of whatever AND/OR are worth. The simplest form is not obvious.

Boundaries condense implication and negation into one single unit BOUNDING, thus merging the languages of Boolean algebra and conventional logic structurally. SHARED SPACE absorbs the duality between AND and OR, leaving forms with clearly countable and comparable complexity metrics.

TWO FORMS OF IFF

The essence of Boolean complexity is that IFF (and its dual XOR) is the only two-valued Boolean function with two different structural forms in the minimalist boundary notation. If it did not have two forms, then all Boolean structures would have straight-line simplification, and the NP=?=P problem would not exist.

Expressed in boundaries, the two forms of IFF are:

```
                    [ [[X] Y] [X [Y]] ]                       IFF-one

                       [X Y] [[X][Y]]                         IFF-two
```

The two forms are necessarily EQUAL:

```
              [ [[X] Y] [[Y] X] ]  =  [X Y] [[X][Y]]          FLEX
```

Square brackets are highlighted parens. Reading for logic, the left-hand-side of the FLEX rule is the conventional form of IFF:

```
           [[[X] Y] [[Y] X]]  -->  (X IMPLIES Y) AND (Y IMPLIES X)
```

The term IF-AND-ONLY-IF derives directly from this representation. The right-hand-side of FLEX has several readings that are not conventional formulations of IFF:

```
              [X Y] [[X][Y]]  -->  (NOT (X OR Y)) OR (X AND Y)

                              -->  (X OR Y) IMPLIES (X AND Y)

                              -->  ((NOT X) OR (NOT Y)) IMPLIES (NOT (X OR Y))
```

The two boundary forms of IFF are of equal complexity. Both have four literals, two in each polarity. IFF-one uses three additional boundaries, while IFF-two uses only two. This asymmetry is swapped in XOR, the negation of IFF, so that the overall boundary count is balanced at 5 per pair of complementary forms. More specifically, the fifth boundary in each pair is an external wrapper that negates the complementary form.

As it turns out, the separation of forms sharing the outermost space in IFF-two outweighs any advantage of having a single top-level form. This advantage is important when determining whether or not two forms, X and Y, are equal, because forms SHARING space are independent. In contrast, an expression in conventional notation must have one and only one top-level connective.

The FLEX rule that equates the two forms of IFF is a special case of boundary DISTRIBUTION:

```
            [ [[X] Y] [[Y] X] ]  =  [X Y] [[X][Y]]          FLEX

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

            with   A = [X Y],   B = X,  and  C = Y.
```

The demonstration applies PERVASION to construct the FLEX equation.

```
            [[[X Y] X][[X Y] Y]]  =  [X Y]  [[X][Y]]              substitute
            [[[  Y] X][[X  ] Y]]  =  [X Y]  [[X][Y]]              pervasion
```

DEFINITION OF EQUALITY

EQUALS can be converted from a Boolean assertion into an assertion about logical consequence, IFF. EQUALS is defined structurally as:

```
      X = Y    =def=    [[[X] Y] [[Y] X]] ==>( )<== [X Y] [[X][Y]]
```

The two structural varieties of IFF are both displayed in the above definition of EQUALS. An arrow, ==>, means that each structural form reduces to Mark by explicit rules (the initials of boundary logic). Each arrow identifies a reduction target, ( ), that defines the validity of the equality; and each arrow implicitly incorporates a set of transformational rules that are accepted as valid transformations. The compound bidirectional joining token

```
                  ==>( )<==
```

is a structural definition of consistency between the two forms of IFF. In the demonstrations below, both forms of IFF are displayed concurrently.

This transformational process lifts the structural constraints of an equation, and converts them into a void-equivalence pattern-matching form. Pattern-matching requires the substitution-of-equals-by-equals mechanism of algebra, but nothing else. Validity is no longer defined by the EQUALS binary connective, it is determined by an identity after "sub-algebraic" pattern-matching.

The definition of EQUALS does not limit the variable X and Y to two-valued Boolean forms. Thus, the conclusions of this paper apply to algebraic forms in general, and are not limited to the interpretation of equality for logic.

REPLICATION

The single rule that differentiates a Boolean algebra from other numerical algebras is REPLICATION, conventionally called IDEMPOTENCY. REPLICATION basically collapses cardinality, so that, in a sense, two is the same as one.

REPLICATION is a special case of PERVASION for which both B and the curly braces are void-equivalent:

```
              A    A      =   A                 REPLICATION

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

PROOF OF IDENTITY

To illustrate the technique of expanding an equation to the boundary form of IFF, consider the case of IDENTITY

```
                  A  =  A                       IDENTITY
```

The proof below converts an algebraic equation into its equivalent structural form by substitution into the definition of IFF. Only the rules of PERVASION and OCCLUSION are then applied to algebraically reduce the IFF forms to Mark, the boundary form of logical TRUE.

```
Prove:            A = A    with    X = A  and  Y = A

      [[[X] Y] [[Y] X]] ==>( )<== [X Y] [[X][Y]]                  def
      [[[A] A] [[A] A]] ==>( )<== [A A] [[A][A]]                  subst
      [[[ ] A] [[ ] A]] ==>( )<== [A  ] [[A]   ]                  -per
      [              ]  = ( )<== [A  ] [[A]   ]                   -occ
      [              ]  = ( )<== [A  ] [      ]                   -per
      [              ]  = ( ) =        [      ]                   -dom
```

The left and right columns are two entirely different and independent proofs, displayed concurrently. They have some reduction steps in common, with the left-hand-side faster to converge on the result. The equivalence of the two forms of IFF is not assumed or used. Each transformation is identified on the far right, with constructive use preceded by "+" and void-equivalent use, by "-".

The last step of the proof uses DOMINION, a trivial theorem of OCCLUSION:

```
                  A ( ) = ( )                        DOMINION

                        A ( )  =?=  ( )
                       (A ( )) =?= (( ))                         +bound
                       (A ( ))  =                                -occ
```

PROOF OF INVOLUTION

The boundary forms of IFF nest each variable, X and Y, at least one level deep. This nesting is the key that resolves several foundational questions in logic, such as the Robbins problem and the role of double negation. Specifically, in the language of boundaries, INVOLUTION is a consequence of the use of EQUALS rather than an initial equation that must be assumed. The definition of EQUALS applies whenever a rule is expressed as an equation. Therefore, since boundary logic is an algebraic system, a sufficient set of initials for OPI is

```
                  (( ) A) =                          OCCLUSION
                  A {A B} = A {B}                    PERVASION
```

INVOLUTION becomes a theorem built implicitly into the use of the EQUALS sign in the algebraic equations.

The two above initials, together with the definition of IFF, can be used to prove INVOLUTION.

```
Prove:            ((A)) = A    with    X = ((A))  and  Y = A

      [[[  X  ] Y] [[Y]   X  ]] ==>( )<== [  X   Y] [[  X  ][Y]]  def
      [[[((A))] A] [[A] ((A))]] =?=( )=?= [((A)) A] [[((A))][A]]      subst
      [[[(( ))] A] [[A] (   )]] =?=( )=?= [(( )) A] [[(   )][A]]      -per
      [[[     ] A] [[A] (   )]] =?=( )=?= [      A] [       [A]]      -occ
      [                       ] =?=( )=?= [      A] [       [A]]      -occ
      [                       ] =?=( )=?= [      A] [         ]       -per
      [                       ]  = ( ) =            [         ]       -dom
```

FUNCTIONAL SUBSTITUTION

One of the mechanisms embedded in algebraic equality is functional substitution:

```
            X = Y   implies   F[X] = F[Y]             FUNCTIONAL SUBSTITUTION
```

In particular for boundary algebra,

```
            X = Y   implies    (X) = (Y)
```

FUNCTIONAL SUBSTITUTION is the first step in the above proof of DOMINION.

The boundary form of logical implication is

```
            X implies Y    =def=     (X) Y
```

FUNCTIONAL SUBSTITUTION of boundaries can be proved rather than assumed, using only the definition of IFF and the two boundary logic rules.

In demonstrating that one equation implies the other, the usual algebraic technique would be to convert one equation into the form of the other, or perhaps to convert both equations into a common third form. The proof below uses the boundary logic technique of converting each equation into a single boundary form using the definition of IFF, and then placing both into the boundary form of implication. This results in one boundary logic form that can then be reduced using only OCCLUSION and PERVASION. The proof technique uses PERVASION to insert an existing form into another form and then to subsequently reduce the result.

FUNCTIONAL SUBSTITUTION expressed purely in boundaries, and without EQUALS, is:

```
    X  =  Y   -->  [ [[ X ] Y ] [[ Y ] X ] ]

   (X) = (Y)  -->  [ [[(X)](Y)] [[(Y)](X)] ]

    X = Y implies (X) = (Y)  -->  ( [[[X] Y] [[Y] X]] ) [[[(X)](Y)] [[(Y)](X)]]
```

Below, FUNCTIONAL SUBSTITUTION is proved twice, once using IFF-one as illustrated above, and once using IFF-two. Forms that are inserted into another form using PERVASION constructively are highlighted by carets, ^...^.

```
Prove:                  X = Y    implies    (X) = (Y)

IFF-one:  [[[X] Y] [[Y] X]]

 ( [[[X] Y] [[Y                         ] X]] ) [[[(X)](Y)] [[(Y)](X)]]  subst
 ( [[[X] Y] [[Y ^[[[(X)](Y)][[(Y)](X)]]^] X]] ) [[[(X)](Y)] [[(Y)](X)]]  +per
 ( [[[X] Y] [[Y ^[[[( )]( )][[( )]( )]]^] X]] ) [[[(X)](Y)] [[(Y)](X)]]  -per
 ( [[[X] Y] [[Y ^[                    ]^] X]] ) [[[(X)](Y)] [[(Y)](X)]]  -occ
 ( [[[X] Y] [                             X]] ) [[[(X)](Y)] [[(Y)](X)]]  -occ
 ( [[    Y] [                             X]] ) [[[(X)](Y)] [[(Y)](X)]]  -per

 ( [[    Y] [X] ^[[[(X)](Y)][[(Y)](X)]]^    ] ) [[[(X)](Y)] [[(Y)](X)]]  +per
 ( [[    Y] [X] ^[[[   ]   ][[   ]   ]]^    ] ) [[[(X)](Y)] [[(Y)](X)]]  -per
 ( [[    Y] [X] ^[                    ]^    ] ) [[[(X)](Y)] [[(Y)](X)]]  -occ
 (                                            ) [[[(X)](Y)] [[(Y)](X)]]  -occ
 (                                            )                          -dom

IFF-two:  [X Y] [[X][Y]]

 ( [X Y               ] [[X][Y]] ) [(X)(Y)] [[(X)][(Y)]]          subst
 ( [X Y               ]          ) [(X)(Y)] [[(X)][(Y)]]          -per
 ( [X Y ^[[(X)][(Y)]]^]          ) [(X)(Y)] [[(X)][(Y)]]          +per
 ( [X Y ^[[( )][( )]]^]          ) [(X)(Y)] [[(X)][(Y)]]          -per
 ( [X Y ^[          ]^]          ) [(X)(Y)] [[(X)][(Y)]]          -occ
 (                               ) [(X)(Y)] [[(X)][(Y)]]          -occ
 (                               )                                -dom
```

IFF-two has two independent forms at the top level. This advantage leads to a shorter proof.

BOUNDARY SUBSTITUTION

INVOLUTION permits the implication of FUNCTIONAL SUBSTITUTION to go both ways, that is, to be IFF:

```
            X = Y    iff     (X) = (Y)                BOUND SUBSTITUTION
```

This proof uses the standard algebraic technique of converting one equation into the other.

```
                  (X)  =  (Y)                                 given
                 ((X)) = ((Y))                                funct subst
                   X   =   Y                                  inv
```

The proof shows that the rule of BOUND SUBSTITUTION is equivalent in power to the rule of INVOLUTION. Given either one, the other is a consequence. Here is a direct algebraic proof of INVOLUTION that uses BOUND SUBSTITUTION:

```
                  (A          )                             lhs
                  (A          ) ((   ))                     +occ
                  (A          ) (((A)))                     +per
                  (A ^(((A)))^) (((A)))                     +per
                  (A ^((( )))^) (((A)))                     -per
                  (A ^(     )^) (((A)))                     -occ
                                (((A)))                     -occ, rhs

                        (A) = (((A)))                       construction
                         A  =  ((A))                        bound subst
```

FUNCTIONAL SUBSTITUTION, proved above, is one direction of the IFF of BOUND SUBSTITUTION. The proof of bidirectionality is completed below by showing that

```
                  (X) = (Y)   implies   X = Y
```

Again, the proof requires only the definition of IFF and the two rules of OCCLUSION and PERVASION. Both varieties of IFF are shown concurrently and independently.

Prove: (X) = (Y) implies X = Y

Variety 1: [[[X] Y] [[Y] X]]

```
 ( [[[(X)](Y)] [[(Y)](X)]] ) [[[X] Y] [[Y] X                            ]]  imp
 ( [[[(X)](Y)] [[(Y)](X)]] ) [[[X] Y] [[Y] X ^([[[(X)](Y)] [[(Y)](X)]])^]]  +per
 ( [[[(X)](Y)] [[(Y)](X)]] ) [[[X] Y] [[Y] X ^([[[( )]   ] [[   ]( )]])^]]  -per
 ( [[[(X)](Y)] [[(Y)](X)]] ) [[[X] Y] [[Y] X ^([[        ]           ])^]]  -occ
 ( [[[(X)](Y)] [[(Y)](X)]] ) [[[X] Y] [[Y] X ^(                       )^]]  -occ
 ( [[[(X)](Y)] [[(Y)](X)]] ) [[[X] Y]                                    ]  -occ

 ( [[[(X)](Y)] [[(Y)](X)]] ) [[[X] Y       ^([[[(X)] (Y)] [[(Y)] (X)]])^]]  +per
 ( [[[(X)](Y)] [[(Y)](X)]] ) [[[X] Y       ^([[[   ] ( )] [[( )]    ]])^]]  -per
 ( [[[(X)](Y)] [[(Y)](X)]] ) [[[X] Y       ^([            [         ]])^]]  -occ
 ( [[[(X)](Y)] [[(Y)](X)]] ) [[[X] Y       ^(                         )^]]  -occ
 ( [[[(X)](Y)] [[(Y)](X)]] ) [                                           ]  -occ
                             [                                           ]  -dom
```

Variety 2: [X Y] [[X][Y]]

```
 ( [(X)(Y)] [[(X)][(Y)]] ) [X Y] [[X][Y]]                         imp
 (          [[(X)][(Y)]] ) [X Y] [[X][Y]]                         -per
 (          [[(X)][(Y)]] ) [X Y] [[X][Y]] [[X][Y]]                +per

 ([[(X)][(Y)]]) [X Y] [[X ^([[(X)][(Y)]])^][Y]] [[X][Y                 ]] format
 ([[(X)][(Y)]]) [X Y] [[X ^([[(X)][(Y)]])^][Y]] [[X][Y ^([[(X)][(Y)]])^]] +per
 ([[(X)][(Y)]]) [X Y] [[X ^([[(X)][   ]])^][Y]] [[X][Y ^([[   ][(Y)]])^]] -per
 ([[(X)][(Y)]]) [X Y] [[X ^(            )^][Y]] [[X][Y ^(            )^]] -occ
 ([[(X)][(Y)]]) [X Y] [                    [Y]] [[X]                    ] -occ

 ([[(X)][(Y)]]) [X Y] [[Y]] [[X              ]]                   format
 ([[(X)][(Y)]]) [X Y] [[Y]] [[X ^[X Y] [[Y]]^]]                   +per
 ([[(X)][(Y)]]) [X Y] [[Y]] [[X ^[  Y] [[Y]]^]]                   -per
 ([[(X)][(Y)]]) [X Y] [[Y]] [[X ^[  Y] [   ]^]]                   -per
 ([[(X)][(Y)]]) [X Y] [[Y]] [                 ]                   -occ
                            [                 ]                   -dom
```

DISTRIBUTION REDUX

We can now verify that DISTRIBUTION can be demonstrated without INVOLUTION:

Prove: A ((B)(C)) = ((A B)(A C))   with X = A ((B)(C)) and Y = ((A B)(A C))

Due to the length of the forms, the two structural varieties of IFF are presented on different lines:

IFF-one:

```
    [[[     X    ]       Y     ] [[     Y      ]      X    ]]] ==> ( )
    [[[A ((B)(C))] ((A B)(A C))] [[((A B)(A C))] A ((B)(C))]]] =?= ( )   subst
    [[[A ((B)(C))] ((A B)(A C))] [[((  B)(  C))] A ((B)(C))]]] =?= ( )   -per
    [[[A ((B)(C))] ((A B)(A C))] [[            ] A ((B)(C))]]] =?= ( )   -per
    [[[A ((B)(C))] ((A B)(A C))]                             ] =?= ( )   -occ

    [[[A ((B)(C)               )] ((A B      )(A C      ))]]   =?= ( )  format
    [[[A ((B)(C) ^((A B)(A C))^)] ((A B      )(A C      ))]]   =?= ( )  +per
    [[[A ((B)(C) ^((  B)(  C))^)] ((A B      )(A C      ))]]   =?= ( )  -per
    [[[A ((B)(C) ^(          )^)] ((A B      )(A C      ))]]   =?= ( )  -per
    [[[A                        ] ((A B      )(A C      ))]]   =?= ( )  -occ
    [[[A                        ] ((A B ^[A]^)(A C ^[A]^))]]   =?= ( )  +per
    [[[A                        ] ((A B ^[ ]^)(A C ^[ ]^))]]   =?= ( )  -per
    [[[A                        ] (                      )]]   =?= ( )  -occ
    [                                                      ]   =?= ( )  -occ
```

IFF-two:

```
    ( ) <== [    X           Y      ] [[    X     ][      Y      ]]
    ( ) <== [A ((B)(C)) ((A B)(A C))] [[A ((B)(C))][((A B)(A C))]]      subst
    ( ) <== [A ((B)(C)) ((  B)(  C))] [[A ((B)(C))][((A B)(A C))]]      -per
    ( ) <== [A ((B)(C))             ] [[A ((B)(C))][((A B)(A C))]]      -per
    ( ) <== [A ((B)(C))             ] [            [((A B)(A C))]]      -per

    ( ) <== [A ((B)(C))] [[((A B               )(A C               ))]] format
    ( ) <== [A ((B)(C))] [[((A B ^[A ((B)(C))]^)(A C ^[A ((B)(C))]^))]] +per
    ( ) <== [A ((B)(C))] [[((A B ^[  (( )(C))]^)(A C ^[  ((B)( ))]^))]] -per
    ( ) <== [A ((B)(C))] [[((A B ^[          ]^)(A C ^[          ]^))]] -occ
    ( ) <== [A ((B)(C))] [[(                                        )]] -occ
    ( ) <== [A ((B)(C))] [                                            ] -occ
                         [                                            ] -dom
```

Thus, DISTRIBUTION and INVOLUTION are both theorems of the two algebraic initials of boundary logic: OCCLUSION and PERVASION.

EVERYTHING BUT INVOLUTION

Kauffman has constructed an algebra system that includes OCCLUSION and PERVASION but not INVOLUTION. The idea is to add another element with a definition that excludes the validity of INVOLUTION. Kauffman's system, called OPe here, is:

```
            A, B in {<void>, ( ), e}

            (A ( )) =                          OCCLUSION
            A {A B} = A {B}                    PERVASION
               (e)  = ( )                      DEFINITION of e
```

OPe excludes INVOLUTION since substitution of e into the form of INVOLUTION results in a contradiction:

```
            ((A))  =?=  A                      INVOLUTION

            ((e))  =?=  e                            subst e
            (( ))  =?=  e                            subst (e)
                   =!=  e                            -occ
```

The definition of e also excludes BOUNDARY SUBSTITUTION. since its use results in a contradiction:

```
               (e)  =  ( )
                e  =!=                               boundary substitution
```

A limited form of INVOLUTION is possible. In all cases:

```
            (((A))) = (A)                      BOUND-INVOLUTION
```

To demonstrate this, only the case of e needs exploration:

```
                  (((e))) =?= (e)
                  ((( ))) =?= ( )                    subst
                  (     )  =  ( )                    -occ
```

Thus the exclusion of INVOLUTION is quite limited, it fails only in the case of ((e)).

DISTRIBUTION FAILS

DISTRIBUTION is a theorem of OPI, and underlies the FLEX rule that equates both varieties of IFF. The proof of DISTRIBUTION requires INVOLUTION, since "A" cannot escape the top-level space without invoking a rule that places it solely in a deeper space. INVOLUTION is the only rule that does so.

```
              A                  ((  B)(  C))
              A                  ((A B)(A C))                     +per
            ((A)               ) ((A B)(A C))                     +inv
            ((A) ^((A B)(A C))^) ((A B)(A C))                     +per
            ((A) ^(          )^) ((A B)(A C))                     -per
                                 ((A B)(A C))                     -occ
```

In the OPI system, INVOLUTION, as well as DISTRIBUTION, can be proved as theorems. In OPe, DISTRIBUTION itself fails in the case of distributing e. The specific violations of the DISTRIBUTION rule can be identified by case analysis.

DISTRIBUTION works in all cases that A ≠ e:

```
B = e              A  (( e )( C )) =?= (( A  e )( A  C ))         subst
                   A  ((   )( C )) =?= (( A  e )( A  C ))         e
                   A               =?= (( A  e )( A  C ))         -occ

  A =                              =?= ((    e )(    C ))         subst
                                   =?= ((      )(    C ))         e
                                    =                             -occ

  A = ( )        ( )               =?= ((( ) e )(( ) C ))         subst
                 ( )                =  (                )         -occ
```

However, e cannot be distributed:

```
                   e  (( B )( C )) =?= (( e  B )( e  C ))

B =                e  ((   )( C )) =?= (( e    )( e  C ))         subst
                   e               =?= (( e    )( e  C ))         -occ
                   e               =?= ((      )( e  C ))         e
                   e               =!=                            -occ

B = e              e  (( e )( C )) =?= (( e  e )( e  C ))         subst
                   e  ((   )( C )) =?= (( e    )( e  C ))         -per
                   e  ((   )( C )) =?= ((      )( e  C ))         e
                   e               =!=                            -occ

B = ( )            e  ((( ))( C )) =?= (( e ( ))( e  C ))         subst
```

```
                 e  (     ( C )) =?= (         (e  C ))         -occ

  C =            e  (     (   )) =?= (        ( e    ))         subst
                 e               =?= (        ( e    ))         -occ
                 e               =!=                            e

  C = ( )        e  (     (( ))) =?= (        ( e ( )))         subst
                 e  (          ) =?= (                )         -occ
                    (          )  =  (                )         -dom

  C = e          e  (     ( e )) =?= (        ( e  e ))         subst
                 e  (     (   )) =?= (        ( e    ))         -per
                 e               =?= (        ( e    ))         -occ
                 e               =!=                            e
```

DISTRIBUTION of e is valid only when both B and C are Mark. This makes sense, since when B and C are both Marked, they dominate each of the spaces being distributed into. In all other cases, e fails to distribute. Through case analysis, the failure of DISTRIBUTION can be seen to be independent of INVOLUTION, since INVOLUTION is never used.

WEAKENED PERVASION

An examination of the steps of the case analysis shows that PERVASION is used in only one circumstance:

```
                e  e  ==>  e                                  -per
```

PERVASION is mandatory to establish DISTRIBUTION, however DISTRIBUTION is not valid in OPe. Thus PERVASION could be weakened; REPLICATION is sufficient to characterize the system OPe.

```
                (A ( )) =                 OCCLUSION
                 A  A   =  A              REPLICATION
                   (e)  = ( )             DEFINITION of e
```

VALIDITY OF THE DEFINITION OF E

The definition of e is itself an equation:

```
            (e) = ( )
```

Therefore we can examine the structure of this definition by substituting into the definition of IFF and then reducing using the other two rules of OPe:

```
Prove:      (e) =?= ( )     with    X = (e)    and    Y = ( )

      [[[ X ]  Y ] [[ Y ]  X ]] ==>( )<== [ X   Y ] [[ X ][ Y ]]
      [[[(e)] ( )] [[( )] (e)]] =?=( )=?= [(e) ( )] [[(e)][( )]]  subst
      [            [     (e)]] =?=( )=?=           [[(e)]    ]  -occ
```

Both versions of IFF yield

```
                  [[(e)]] =?= ( )                    (e)-INVOLUTION
```

That is to say, for the definition of e to be valid, INVOLUTION must apply to (e), which it does as shown earlier in the section.

But there is a problem: it is not possible to define e to be different than the ground value <void> in OPe. That is, two of the elements in OPe are not themselves unique, thus violating the equational presumption of unique labeling of ground forms.

Here we want the equality tests to fail:

```
Prove:      e =?=  <void>        with    X = e    and    Y = <void>

      [[[ X ]  Y ] [[ Y ]  X ]] ==>( )<== [ X   Y ] [[ X ][ Y ]]
      [[[ e ]    ] [[   ]  e ]] =?=( )=?= [ e     ] [[ e ][   ]]  subst
      [[[ e ]    ]            ] =?=( )=?= [ e     ]                     -occ
      [[[   ]    ]            ] =?=( )=?= [       ]                     e
      [                       ]  = ( ) =  [       ]                     -occ
```

Given the definition of e, it cannot be differentiated from <void>. For the system OPe, we lose both DISTRIBUTION and INVOLUTION, and the source of this is failure of the uniqueness of ground elements. That is, the OPe basis set is both:

```
            {<void>, ( ), e}  and  {<void>, ( )}
```

SUMMARY

We have used three ideas to show that the definition of e is not unique.

1) REPLICATION and OCCLUSION as the only rules
2) the structural definition of IFF
3) the conventional transformation techniques of algebra.

INVOLUTION is not the central issue for OPe. It fails as a rule only as a by-product of the non-uniqueness of the elements. A possible lesson is that elements must be defined to be unique de novo, rather than indirectly by equations such as

$$(e) = (\ ) \qquad\qquad \text{DEFINITION of } e$$

If e were unique, then the definitional equation above would not be valid. When we apply an algebraic operation such as FUNCTIONAL SUBSTITUTION, the operation needs to be consistent. This is possible only if the basis elements are unique. Alternatively, we can accept the definition at the cost of undermining the definition of the EQUALS sign.

EQUALITY IN THE ARITHMETIC

Spencer-Brown's book, Laws of Form, introduces two initial equations, CALLING and CROSSING, that define an arithmetic of boundaries.

```
                  ( ) ( )  =  ( )              CALLING

                  ( ( ) )  =                   CROSSING
```

The initials consist only of boundaries (no variables, thus an arithmetic) and the EQUALS sign. EQUALS is defined as a type of indistinguishability, "is confused with".

The theme is that EQUALS is not free, it constrains and defines the initial axioms of a system. How does the algebraic format of this system interact with the content of the initial arithmetic equations? Specifically, is one of the rules implicitly built into the EQUALS sign used by the other?

We use the definition of IFF to show that this is indeed the case. CROSSING is a theorem of a strengthened variety of CALLING: arithmetic PERVASION.

ARITHMETIC PERVASION

PERVASION, which plays such a central role in the algebra, has the following form in the arithmetic:

```
            ( ) {( )}  =  ( ) {   }            arithmetic PERVASION-1
```

Again, the curly braces stand in place of any number of inward facing boundaries with respect to the pervading, outside form.

When the curly braces stand in place of no boundaries, PERVASION becomes CALLING; thus, PERVASION therefore extends CALLING into deeper spaces.

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

Using the same technique of appealing to the structural definition of EQUALITY, can we show that arithmetic PERVASION is sufficient as a single initial?

The reduction of nested boundaries appears to pose an immediate problem for PERVASION as it is minimally defined above. In an arithmetic, rules cannot be generalized to arbitrary configurations, thus the following simple example does not reduce by either CALLING or PERVASION.

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

Again, it appears as though CROSSING is mandatory.

To be effective as a single initial, arithmetic PERVASION must be extended to include a second case:

```
          (( )) {(( ))}  =  (( )) {   }             arithmetic PERVASION-2
```

This extension is not the same as including the rule of CROSSING, since the remaining double boundary cannot be equated to <void>. The following reduction, however, is enabled:

```
               (( )) ( (( )) )                   form
               (( )) (       )                   -per2
               (   ) (       )                   -per1
               (   )                             -per1
```

IDENTITY

Consider first the case of identity:

```
Prove:         ( ) = ( )     with    X = ( )  and  Y = ( )

     [[[ X ]  Y ] [[ Y ]  X ]] ==>( )<== [ X   Y ] [[ X ][ Y ]]
     [[[( )] ( )] [[( )] ( )]] =?=( )=?= [( ) ( )] [[( )][( )]]  subst
```

By observation, it is apparent that both of the arithmetic initials are needed to reduce these forms. Use of CALLING leaves (( )) forms that cannot be further reduced. Use of CROSSING leaves ( ) ( ) forms that cannot be further reduced.

The two cases of PERVASION, however, are sufficient to demonstrate the identity of Mark:

```
     [[[( )] ( )] [[( )] ( )]] =?=( )=?= [( ) ( )] [[( )][( )]]  copy
     [[[   ] ( )] [[   ] ( )]] =?=( )=?= [( )    ] [[( )][( )]]  -per1
     [[[   ]    ] [[   ]    ]] =?=( )=?= [( )    ] [[( )][( )]]  -per1
     [[[   ]    ]             ] =?=( )=?= [( )    ] [          ]  -per2
     [[[   ]    ]             ] =?=( )=?= [        ] [          ]  -per1
     [[[   ]    ]             ] =?=( )=?= [        ]              -per1
```

What is interesting here is a central theme of this section: arithmetic PERVASION is sufficient for only one of the varieties of IFF.

In boundary algebra, there is an even simpler identity:

```
Prove:      <void> = <void>     with    X = <void>  and  Y = <void>
```

```
      [[[X] Y] [[Y] X]] ==>( )<== [X Y] [[X][Y]]
      [[[ ]  ] [[ ]  ]] =?=( )=?= [   ] [[ ][ ]]                  subst
      [[[ ]  ] [[ ]  ]] =?=( )=?= [   ] [      ]                  -per1
      [[[ ]  ] [[ ]  ]] =?=( )=?= [   ]                           -per1
      [[[ ]  ]        ] =?=( )=?= [   ]                           -per2
```

Again, PERVASION is sufficient for the right-hand variety of IFF, but not for the more conventional left-hand variety.

COMPUTATION

CALLING extends the cardinality of syntactic Marks into a SHARING space, while CROSSING extends the ordinality of syntactic Marks into BOUNDED space. The two extensions (inside/outside, cardinal/ordinal, SHARING/BOUNDING) are orthogonal by construction. The "breadth" space constructed by CALLING is independent of the "depth" space constructed by CROSSING.

PERVASION generalizes CALLING to incorporate BOUNDED depth as well as SHARING breadth; both can be transformed by the single rule. PERVASION renders inward facing parens transparent, maintaining structure solely by the containment of inner spaces.

Computational implementations are algebraic to improve efficiency, transformations within the arithmetic are quickly obscured and excessively expensive. In the two sets of examples below, a computational implementation would trigger the appropriate algebraic rule, yielding the result in one step.

```
      CALL/CROSS               arithmetic-PERVASION     algebraic reduction

   ( ) (( ) ( ))               ( ) (( ) ( ))              ( ) (( ) ( ))
   ( ) (( )    )     call      ( ) (       )     per1     ( )              dom
   ( )               cross     ( )                 per1


      CALL/CROSS               arithmetic-PERVASION     algebraic reduction

  (( ) ((((( ))))))            (( ) ((((( ))))))         (( ) ((((( ))))))
  (( ) (((     ))))  cross     (( ) ((((   )))))  per1                     occ
  (( ) (         ))  cross     (( ) (((     ))))  per1
  (( )            )  call      (( ) ((       )))  per1
  (( )            )  cross     (( ) (         ))  per1
                               (( )            )  per1
                                                  cross
```

Disregarding computational efficiency, the arithmetic of CALLING and CROSSING has a structural elegance. The notation defines two different processes that are applied to two different types of structure. The two orthogonal rules permit two independent reduction regimes. Forms SHARING space are in parallel, nested parens require sequential steps.

PERVASION, however, unifies breadth and depth of form into a single "spatial" transformation. Viewed in a parallel implementation, PERVASION can extract any and all inner Marks in the presence of any external Mark, regardless of nesting.

The bottom line is that *notation* should never exclude a possible implementation of the transformation being described. When a set of transformation rules is applied in a *single algorithmic step*, the multiplicity of the rules themselves is motivated by cognitive rather than computational objectives. In a machine implementation, PERVASION is a single algorithm; it is not necessary to decompose it into CALLING and a type of CROSSING.

BUILDING CALLING

CALLING is a subcase of PERVASION, and can be shown to be consistent:

```
Prove:        ( ) ( ) = ( )   with   X = ( ) ( )   and   Y = ( )

  [[[   X  ]  Y ] [[ Y ]    X  ]] ==>( )<== [   X    Y ] [[   X  ][ Y ]]
  [[[( )( )] ( )] [[( )] ( )( )]] =?=( )=?= [( )( ) ( )] [[( )( )][( )]]   subst
  [[[      ] ( )] [[   ] ( )   ]] =?=( )=?= [( )       ] [[( )   ][( )]]   per1
  [[[      ]    ] [[   ]       ]] =?=( )=?= [( )       ] [[( )   ][( )]]   per1
  [[[      ]    ]                ] =?=( )=?= [          ] [             ]   per2
  [[[      ]    ]                ] =?=( ) =  [          ]                   per1
```

BUILDING CROSSING

Since PERVASION incorporates CALLING as a subcase, the primary question is whether or not CROSSING is essential to the initials. From the arithmetic PERVASION and the definition of EQUALS, we can construct/prove CROSSING?

```
Prove:       (( )) =  <void>        with  X = (( ))  and  Y = <void>

    [[[  X  ] Y] [[Y]   X  ]] ==>( )<== [  X   Y] [[  X  ][Y]]
    [[[(( ))]  ] [[ ] (( ))]] =?=( )=?= [(( ))  ] [[(( ))][ ]]        subst
    [[[(( ))]  ] [[ ] (   )]] =?=( )=?= [(( ))  ] [[(   )][ ]]       -per1
    [[[(( ))]  ] [[ ]      ]] =?=( )=?= [(( ))  ] [[     ][ ]]       -per1
    [[[(( ))]  ] [[ ]      ]] =?=( )=?= [(( ))  ] [[     ]  ]       -per1
    [[[     ]  ] [[ ]      ]] =?=( )=?= [       ] [[     ]  ]       -per2
[[[     ]  ]            ] =?=( )=?= [       ] [[     ]   ]        -per2
[[[     ]  ]            ] =?=( )=?= [       ] [          ]        -per1
[[[     ]  ]            ] =?=( ) =  [       ]                     -per1
```

Again we see the characteristic pattern that the proof completes with IFF-two but not with IFF-one.

It is very interesting that the FLEX transformation (a special case of DISTRIBUTION) can alter the course of the right-hand reduction to avoid the necessity of INVOLUTION. If FLEX is assumed, this variance does not occur, since steps from the right-hand conclusion can be traced upward through the definition of FLEX and down to the right-hand conclusion, yielding a proof for a subcase of INVOLUTION:

```
            ((( )))  =  ( )               BOUNDED CROSSING
```

THE STRANGE RESULT

There is a subtle interaction between the arithmetic rules and the use of the equality symbol. Specifically, the non-conventional form of IFF, IFF-two, that consists of two separate subforms reduces using only the two cases of arithmetic PERVASION, while the conventional form of IFF, IFF-one, that is the conjunction of two implications becomes too deeply nested to reduce.

The question being pursued is whether or not a basis set of rules for boundary algebra needs INVOLUTION. That is, can INVOLUTION be shown to be a theorem of the other rules? The algebraic result is that INVOLUTION is built into the equality sign, and as such is assumed in the notion of *algebra*.

In the arithmetic, the same can be said for CROSSING, that it is built into the equality token introduced in the two cases of arithmetic PERVASION.
The role of INVOLUTION at the basis of logic is confounded with the structure of the algebraic equality statement. IFF is the source of Boolean complexity, in that it has two structural definitions. Without *both* of these definitions, we cannot engage in algebraic transformation. The equality of the two structural definitions embeds the rule of DISTRIBUTION within the algebraic process. DISTRIBUTION, in turn, requires INVOLUTION, PERVASION and OCCLUSION. However, INVOLUTION can be shown to be a consequence of IFF and PERVASION.